TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

Particle-Based Fluid Visualisation on the GPU

By
Kees van Kooten

Supervisors:

Alex Telea (TU/e)
Gino van den Bergen (Playlogic)
Herman Haverkort (TU/e)

Eindhoven, August 2006

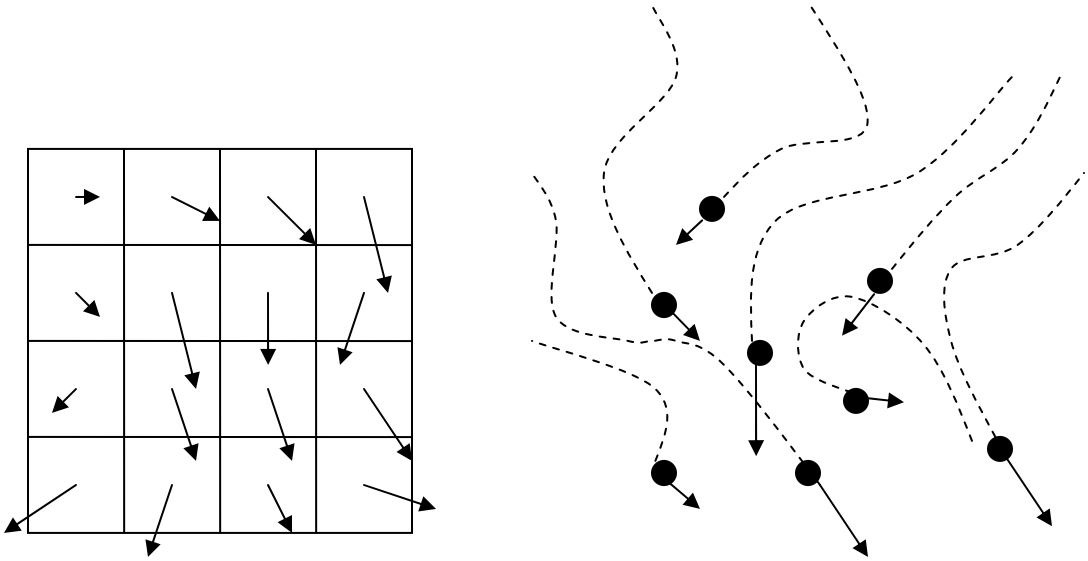
Introduction

1.1 An introduction to Fluid Simulation

Fluid simulation and visualisation is becoming an area of increasing interest for interactive applications, especially since the arrival of consumer-level programmable hardware for graphics acceleration. This stems from the desire to model virtual environments with rigid as well as deformable fluid-like bodies with increasing levels of realism, while maintaining the interactivity of the application. Games and virtual environments are typical examples of situations in which a realistic-but-efficient fluid simulation is useful.

In general, there are two different methods for simulation of a fluid: Eulerian and Lagrangian. Eulerian simulations discretise a given space into a fixed number of cells. Typically, the discretisation is uniform, involving a regular lattice of cells of equal size. For every cell, several quantities are computed, such as velocity, density, and pressure, which describe the fluid flow in that spatial extent. While these quantities change over time, the spatial discretisation does not. Using the Navier-Stokes equations discretised on a fixed grid which describe the flow of fluids, changes in quantities from a single cell can be determined by examining the state of neighbouring cells. Lagrangian methods take a different approach. Instead of creating a fixed discretisation of space, these methods keep track of fluid attributes only at the positions of moving particles; the fluid 'atoms'. The particles 'carry' attributes like size, mass, position, velocity, and pressure. Changes in these parameters are therefore considered following the trajectory of a fluid particle. An example of such a parameter is the fluid density: by keeping track of the density at every particle's position, this value can be interpolated at every other position in the fluid. The resulting field of values can then be used to derive forces on every particle, from which the movement of the fluid is calculated. This in turn changes the density at particle positions again. The difference between the fixed-position Eulerian approach and the dynamic Lagrangian approach is highlighted once again in [Figure 1].

Eulerian and Lagrangian simulation methods have their own advantages and limitations. Eulerian methods are relatively simpler to implement, since they assume a fixed spatial discretisation. Also, they allow one to control explicitly where one has information about the flow, by controlling the placement of the spatial cells. The fixed property of these cells can also be a disadvantage, for it implicitly fixes the location of the fluid itself; fluids cannot be simulated at positions outside of the discretised volume. On top of this, the explicit volumetric discretisation has to be stored somewhere, which can be very expensive for complex and accurate simulations. Lagrangian methods are relatively cheaper. Since one explicitly controls the positions of the flow particles, one can easily achieve highly non-uniform sampling of a flow phenomenon. In simple terms, one can control the spatial particle distribution, thus specify where one wants to have more flow detail. However, this advantage comes together with a disadvantage: the density of the particles must be explicitly controlled in order to avoid extreme sub-sampling of certain areas on one hand, as this yields inexact solutions, and on the other hand to avoid extreme super-sampling of other areas, as this increases computational times unnecessarily.



The left figure shows the grid-based Eulerian approach to fluid simulation: every grid cell contains a flow vector denoting the flow of the fluid through the static cell. The right figure shows the Lagrangian approach: black dots depict fluid particles moving freely through space, with a dashed line as their trajectory and an arrow as the velocity vector which is considered along the corresponding trajectory.

Figure 1

Both methods have applications ranging from very crude approximations used as visual effects in software music players to highly accurate scientific simulations for research purposes. The algorithms described in this document are developed for interactive simulations in games and virtual environments. A particle-based method called Smoothed Particle Hydrodynamics (SPH) has recently been proposed by [I], aimed specifically at interactive applications. This method will be used to provide the basis for the fluid simulation used for visualisation, and it is treated in Section 2 of this work.

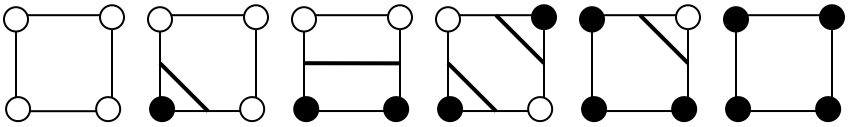
While [XII] and [IV] discuss implementations on the graphics hardware of both grid-based and particle-based fluids respectively, this is not a requirement for the SPH method in this work; the simulation will run on commodity consumer processors. Section [1.3] provides more detail on the mapping of the simulation and visualisation to the available hardware.

1.2 An introduction to Fluid Visualisation

After the fluid flow has been simulated, it can be visualised. Visualisation of a fluid simulation is the main topic of this work. Just as for the simulation phase, several options are available here. We could visualise the fluid flow using one of the various flow visualization techniques well-known in scientific visualisation, e.g. vector plots, glyphs, stream surfaces, streamlines, or texture-based visualization. An overview of these techniques is given in [XIII]. In this work, we are interested in visualizing the surface of a

3D fluid. One of the possible ways to accomplish this has been mentioned in [XIII]: isosurface extraction with Marching Cubes, introduced in [XIV]. Another way to visualise the surface is to employ Surface Particles, as in [II]. While both visualisation methods can be applied to any type of fluid simulation, they differ in their methodology: Marching Cubes is a fixed grid-based approach, while the surface particles method is – as the name suggests – particle-based. A third approach for surface visualisation is ray-tracing, but this is not a feasible solution due to hardware and performance requirements discussed later in this section.

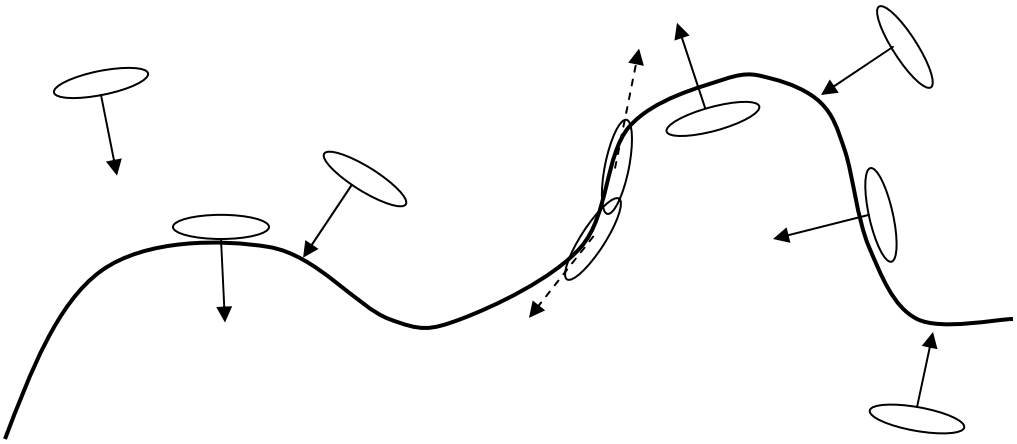
The Marching Cubes algorithm discretises a 3D volume into cubical cells. At every corner of a cell, the value of the field quantity – e.g. the fluid density – is determined from which an isosurface needs to be extracted. Based on this value, a corner is marked to be inside or outside the fluid. The Marching Cubes algorithm then searches for cells which the fluid surface intersects, i.e. the cells with corners located both inside and outside the fluid. Based on the configuration of the corners inside and outside the fluid, a fluid surface can be interpolated. For the 2D case, this is demonstrated in [Figure 2]. The 3D case is similar, but instead of lines intersecting squares, one constructs triangles intersecting cubical cells.



Multiple variants of a single cell after discretisation by the Marching Cubes algorithm. The corners are determined to be inside or outside the fluid volume, depicted by black and white dots respectively. For every possible combination, the fluid surface can be interpolated within the cell, which is visible by the thick black line. All combinations are listed here, except for their rotated variants.

Figure 2

The Surface Particles technique simulates a set of particles in 3D space in order to move them to the surface of the fluid. Based on a particle position, the gradient of the quantity field can be established, along which the particle moves into the direction of the fluid surface. This is how particles independently ‘find’ a place on the surface of the fluid. Particles located close to each other produce repulsion forces, to distribute themselves evenly across this surface. The idea is illustrated once again for the 2D case in [Figure 3].



Surface particles on a fluid surface. The black curve represents the fluid surface, while the surface particles are white ellipses. The arrows denote movement of a particle in a particular direction. Particles can move in the direction of the gradient of the fluid – denoted by black arrows – to find the fluid surface. Particles on the fluid surface can repel each other when they are too close. This is illustrated by dashed arrows.

Figure 3

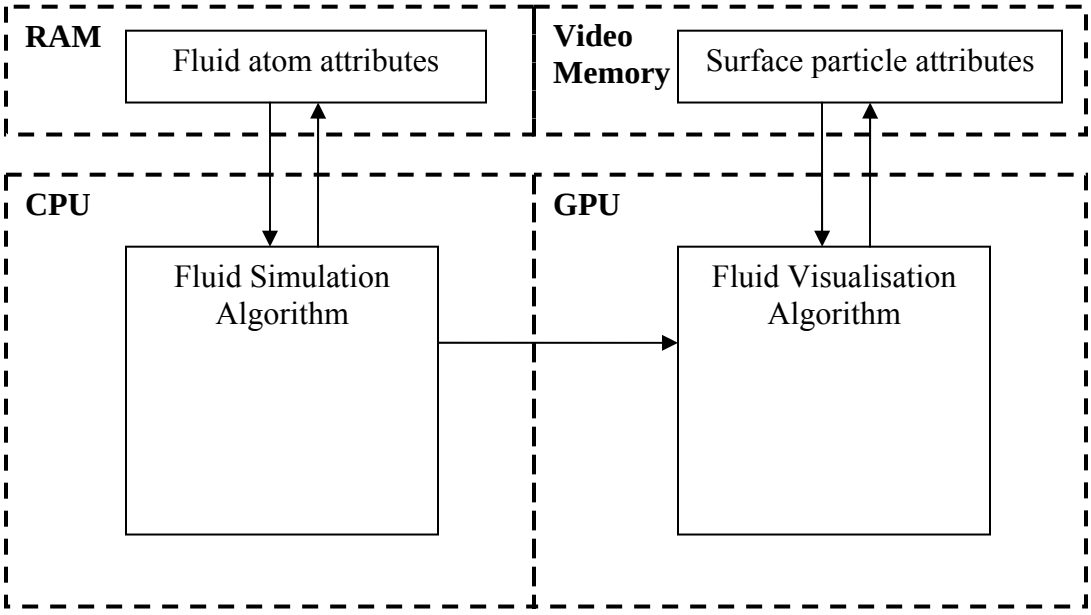
The advantages and limitations of Marching Cubes and Surface Particles are very similar to those of Eulerian and Lagrangian fluid simulation methods. Marching Cubes is easier to implement but has as a disadvantage the cost of scanning through a cubic volume to obtain a surface. This cost can however be reduced to quadratic complexity, putting it much closer to a Langrangian method in terms of performance. On the other hand, the method of Surface Particles is in essence cheaper, but the effort to maintain an even distribution of particles on the fluid surface complicates the algorithm. In practice, it is therefore possible that the performance difference between Marching Cubes and Surface Particles is much smaller than between Eulerian and Lagrangian fluid simulation methods.

Our goal will to strike an optimal balance between quality of visualisation and computational complexity. The visualisation does not have to be accurate, but the effect should be convincing in displaying the complete fluid surface. On top of this requirement comes a second one: instead of visualising the fluid surface with the general computational model of consumer computers, the visualisation algorithm is required to run solely on commodity graphics hardware. Section [1.3] provides more information on the mapping to this hardware.

While fluid visualisation on 2D surfaces has already established itself on graphics hardware by [XII], the development of fluid surface visualisation in 3D has not progressed that much. An implementation of the Marching Cubes algorithm on graphics hardware is given in [XV], but has as disadvantage that the algorithm evaluates the complete cubic fluid volume. An implementation of Surface Particles on graphics hardware has been absent so far; particle-based visualisation of fluids commonly consists of placing particles at atom positions instead of locations on the fluid surface.

1.3 Problem Statement

As mentioned in sections [1.1] and [1.2], the aim of this work is to construct a convincing fluid visualisation performing at interactive speeds on common graphics hardware. The graphics hardware consists of a Graphics Processing Unit (GPU), with Video Memory. As the graphics hardware will be stressed by the visualisation algorithm, the fluid simulation itself will be executed on the Central Processing Unit (CPU) with available Random Access Memory (RAM). An example of what is meant by commodity hardware, is a 2 Ghz AMD or Intel CPU with nVidia GeForce 6800 video card better, supporting Shader Model 3.0. The sketch of [Figure 4] clarifies the mapping of the different software components on the hardware:



The mapping of the software components on the hardware. Arrows denote read/write dependencies. A more detailed view is given in [3.6.1].

Figure 4

The information transported between the fluid simulation and visualisation consists of the fluid atom data required for reconstructing the fluid surface at any position. Examples of this data are fluid atom positions, velocities and densities. The rationale behind the choice of transported data, coupled with a detailed schematic can be found in section [3.5] and [3.6.1].

While the described visualisation techniques are applicable to both a Eulerian as well as a Lagrangian fluid simulation, the focus will be entirely on a particle-based Lagrangian fluid simulation method called Smoothed Particle Hydrodynamics. While the choice of simulation method is not part of this work, two supporting arguments can be found. First, it is largely due to the n^3-size nature of the grid in Eulerian methods, resulting in large performance and memory requirements for real-time applications. The consequence of

this will be a high amount of data being transported between the simulation and visualisation. This is a limiting factor – especially in our case where the simulation and visualisation are performed on different hardware. The second reason to choose a particle-based fluid simulation is ease of animation. Particles lend themselves to animation of highly deformable bodies much easier than a grid-based solution. This is desirable for applications in games and movies, where fluid-like objects have to be created that do not necessarily behave according to the laws of fluid dynamics.

For visualisation, we choose to use a particle-based approach as well. The main reason for this is the absence of an implementation of this algorithm on the GPU. Finding a way to do this, evaluating the performance and the distribution of particles on the fluid surface is interesting research material. Another good reason is speed of execution: particles can exploit the temporal coherence of the fluid surface. The position of a particle can be determined by its previous position and the motion of the fluid atoms. Calculating an updated position for every particle this way is much cheaper than scanning the complete 3D space every frame in order to determine where the surface of the fluid resides. Updating the attributes of particles will ideally be less expensive than scanning the complete fluid volume to identify the fluid surface. The latter is akin to "starting from scratch" each and every frame, while the former can be described as "tweaking the existing solution". The last factor contributing to the choice for a particle-based visualisation, is the adaptability of a particle-based solution: particles can easily be changed in size, number and look. This makes the solution very suitable to variations in level-of-detail.

Note that the simulation of surface particles performed by the visualisation algorithm is different from the simulation of fluid atoms; the latter are often called particles as well in existing particle-based fluid simulation techniques. In the following text, the fluid atoms comprising the fluid itself are always referred to as "atoms" instead of "particles". The term "particles" refers to the particles moving around on the surface of the fluid.

As noted before, a particle-based visualisation running on graphics hardware is not readily available. Therefore, a concise description of the problem treated in this work, is to find a method to visualise the fluid surface of a particle-based fluid simulation, by using particles for the visualisation and simulating their movement on the GPU.

1.4 Contributions

Contributions are presented for the simulation of the surface particles during visualisation, and this can be separated into three parts.

The first part consists of a solution to efficiently constrain the particles to the surface of the fluid, such that every particle represents a section of the surface. Specifically, a data structure is chosen in order to maximise the efficiency of simulating the surface particles on the GPU, based on the required data provided by the fluid simulation. The choice is outlined in [3.6.2].

In itself this will not be sufficient to visualise the fluid surface, because there is nothing to prevent parts of the fluid surface to not be represented at all, or to be represented more than once due to overlapping particles. Therefore, the second part of the simulation entails a solution to the particle distribution problem; an algorithm is developed to uniformly distribute the particles on the fluid surface over the area of this surface. The second and third contributions are both a component of the particle distribution algorithm; the second contribution locally distributes particles based on repulsion forces, while the third contribution accelerates the distribution by globally distributing particles over the full fluid surface based on surface density differences. The concept of the local distribution algorithm is presented in [3.3.2], with its crucial component being the actual method of implementation, described in [3.6.4]. The concept of the global distribution algorithm is presented in [3.3.3], with its implementation based on the method of [3.6.4] as well.

Our main focus is designing an effective and efficient method for visualizing fluids. Thereby we aim to improve in the areas of fluid and fluid surface representation methods and computation algorithms. We do not aim to improve in the areas of actual fluid surface rendering. For this task, we shall use standard particle rendering methods, and not treat the more extensive topics of high-quality particle blending methods, transparency, volumetric effects, and surface detail rendering.

1.5 Structure of the Essay

This essay is structured as follows: in section 2, a particle-based fluid simulation technique called Smoothed Particle Hydrodynamics (SPH) will be described. After an introduction in [2.1], the abstract model of fluid dynamics equations is presented by [2.2]. This introduces a set of fluid field quantities, which are interpolated by SPH. The interpolation is described in [2.3]. Finally, section [2.4] presents a computational way to solve the system of equations relating to the field quantities. This will help in determining the quantities required for visualisation.

Section 3 presents the method of visualisation. An introduction by [3.1] precedes the problem statement in [3.2]. An abstract solution to the problem is posed by [3.3]. Again, quantities are introduced which are only known at discrete particle locations; the interpolation of these quantities is treated in [3.4]. Section [3.5] constructs a system of equations for the fluid visualisation algorithm. In this section, the system of equations is also related to the fluid simulation system presented in section [2.4]. After the theoretical model has been established, section [3.6] provides implementation details. That includes the mapping of the fluid simulation and visualisation to the hardware, and implementation details for both data transport between the two components and particle distribution during visualisation. The characteristics of the implementation are tested in section [3.7], providing results for both performance and visual appearance. Finally, [3.8] concludes the fluid visualisation section by suggesting improvements upon the established algorithms.

2 Fluid Simulation with Smoothed Particle Hydrodynamics

2.1 Introduction

An overview of the fluid simulation before explaining the technique for visualising the fluid surface will be helpful in two ways. First, the relation of input-output between the two systems can be established, giving insight into data transport requirements when mapping them to different hardware. This is described in detail by section [3.5.1]. Secondly, the technique of interpolation for fluid simulation – SPH – is used for distribution of particles in the fluid visualisation as well. This will become evident in section [3.4.2].

The description of the fluid simulation will mainly cover the attributes of the fluid atoms and the formulas with the parameters used for deriving these attributes. For a more detailed description of the technique, see [I].

First, an abstract overview of particle-based fluid simulation will be given in [2.2]. Following the abstract model will be the interpolation with Smoothed Particle Hydrodynamics in [2.3], such that the theoretical particle-based model can be simulated on a computer. Finally, the steps of the simulation model are put in order together with their corresponding inputs and outputs by [2.4], to provide a full overview of the simulation algorithm and to define the input/output relation with the visualisation algorithm.

2.2 Abstract Model

2.2.1 Particle-Based Fluid Simulation

Particle-based fluid simulation is based on representing a fluid with a possibly infinite amount of infinitely small fluid atoms of constant mass moving through space. The consequence of the ability to move is that each fluid atom is governed by the classical equations of motion:

$$\boldsymbol{x}_t = \boldsymbol{x}_0 + \boldsymbol{v}_0 \cdot t + \frac{1}{2} \cdot \boldsymbol{a}_0 \cdot t^2$$

Equation 1

and

$$\boldsymbol{v}_t = \boldsymbol{v}_0 + \boldsymbol{a}_0 \cdot t$$

Equation 2

for a time range [0..t], with position $\boldsymbol{x}$, velocity $\boldsymbol{v}$ and acceleration $\boldsymbol{a}$ with subscripts denoting time. Bold symbols signify a vector, while non-bold symbols are scalars. So the velocity and position of a fluid atom at time t can be calculated by integrating the acceleration over time.

The determining factor for fluid-like behaviour of the atoms stems from the way in which the atoms' accelerations are calculated. In the case of fluids, acceleration is part of the Navier-Stokes equation for conservation of momentum. As explained in [I], this equation can be simplified considerably for particle-based fluids, yielding the following equation governing the acceleration of a fluid atom:

$$\boldsymbol{a}(\boldsymbol{x}_t) = \frac{\boldsymbol{f}(\boldsymbol{x}_t)}{\rho(\boldsymbol{x}_t)}$$

Equation 3

where $\boldsymbol{f(x_t)}$ is the force density field acting on the atom's position $\boldsymbol{x}_t$ and ρ the density field value of the fluid at that position. The density ρ of a fluid is expressed in mass per unit volume, while force density $\boldsymbol{f(x_t)}$ consists of three components, taken from [I]: $\boldsymbol{f}_{pressure}$, $\boldsymbol{f}_{viscosity}$ and $\boldsymbol{f}_{external}$.

The first component, $\boldsymbol{f}_{pressure}$, is the gradient of the pressure field. The pressure p is linearly dependent on the density, with an equation similar to the ideal gas law, proposed by [IX]:

$$p(\boldsymbol{x}_t) = k(\rho(\boldsymbol{x}_t) - \rho_0)$$

Equation 4

where ρ_0 is the rest density constant, in most cases chosen to be close to 0. The rest density has no mathematical influence on the gradient of the pressure field. However, it does have an influence on the gradient of the pressure field after discretisation with SPH, and it can also be used to improve numerical stability. In effect, the rest density will be used as a "guideline" for $\boldsymbol{f}_{pressure}$, such that this force will always be directed towards the rest density. For now however, $\boldsymbol{f}_{pressure}$ is defined as:

$$\boldsymbol{f}_{pressure}(\boldsymbol{x}_t) = -\nabla p(\boldsymbol{x}_t)$$

Equation 5

which is the negative gradient of the pressure at position $\boldsymbol{x}_t$.

The second component, $\boldsymbol{f}_{viscosity}$, is defined by the following term, according to [I]:

$$\boldsymbol{f}_{viscosity}(\boldsymbol{x}_t) = \mu \cdot \nabla^2 \boldsymbol{v}(\boldsymbol{x}_t)$$

Equation 6

where μ is the viscosity constant and $\boldsymbol{v}$ the velocity of the fluid. Intuitively, $\boldsymbol{f}_{viscosity}$ points into the direction of the velocity gradient, counteracting changes in velocity with respect to the environment of position $\boldsymbol{x}_t$.

Apart from the previous two internal force components, one could choose to define others as well. One of the possible options is to model the surface tension of the fluid. This is explained in detail in [XI], while some of the ideas presented there return in [I].

The last component of the force density, $\boldsymbol{f}_{external}$, consists of external forces and is therefore specific for each different fluid simulation. One could model fluid in a glass subjected to collision forces with the glass just as well as experimenting with gravitational fields, requiring different models for the external forces.

In conclusion, the dynamic system of a particle-based fluid can be modelled with the parameters $\boldsymbol{f}$ and ρ at every position in space, yielding an "acceleration field" that can be evaluated at the position of every fluid atom. The result can be used to integrate the velocity $\boldsymbol{v}$ and position $\boldsymbol{x}$ of the fluid atoms forward in time.

2.3 Interpolation of Field Quantities

2.3.1 Smoothing Kernels

While the section [2.2] established a model for simulating the movement of a particle-based fluid, this model does not translate directly onto a computer with only finite memory and calculation time. Only a small amount of fluid atoms are simulated, but [Equation 3], [Equation 5] and [Equation 6] dictate that field quantities ρ, $\boldsymbol{f}_{pressure}$ and $\boldsymbol{f}_{viscosity}$ are known at every location within the fluid. Although every fluid atom keeps the attributes velocity, position and mass, this is in itself insufficient to determine the aforementioned quantities around its central position. The only available information is the contribution of the atom to the quantity at its central position, and not its surrounding environment. Since this environment is discretised by only a few fluid atoms, a quantity like density can therefore not directly be extracted for any position. After all, not every particle has the same position, so at every position there are particles from which the contribution to the density field is unknown. As a result, the value of a quantity like fluid density is unknown for positions around a fluid atom. To solve this problem, the contribution of a fluid atom towards a quantity at locations around its position is interpolated. Through establishing a method of interpolation, quantities ρ, $\boldsymbol{f}_{pressure}$ and $\boldsymbol{f}_{viscosity}$ can be calculated at every position, including the fluid atom positions.

The specific method of choice for interpolating the particle-based fluid simulation model is Smoothed Particle Hydrodynamics, which was introduced in [X] and has also been discussed in [I] and [IX]. This is a method for interpolating field quantities at arbitrary discrete locations while only having knowledge of the values of these quantities at the positions of a finite number of fluid atoms. Examples of quantities in the case of fluid

simulation are the density from [Equation 3] or the forces from [Equation 5] and [Equation 6]. The method of interpolation of these field quantities is based on a single principle, which can be applied to every parameter of the fluid. Therefore, the SPH technique provides a simple method of calculation. Furthermore, the possibility of limiting the number of fluid atoms to a small amount will make the simulation tractable for real-time applications.

To see how the SPH method is applicable to the abstract fluid simulation model, consider the acceleration for every fluid atom used to integrate positions and velocities of fluid atoms over time. According to [Equation 3], the acceleration is dependent on the evaluation of a force density field and a density field, to obtain an $\boldsymbol{f}$ and ρ at any position. As noted, the calculation of these field quantities is performed by interpolation according to the SPH principle, after which the acceleration of the fluid atom can be determined.

Smoothed Particle Hydrodynamics is distinctive in its approach towards interpolating aforementioned field quantities by making use of a concept called Smoothing Kernels. A Smoothing Kernel is a function

$$W(\boldsymbol{r},h) \quad :\Re^3 x\Re \rightarrow \Re$$

Equation 7

for a location $\boldsymbol{r}$ and core radius h.

Intuitively, a field quantity will be interpolated at position $\boldsymbol{r}$ by placing a Smoothing Kernel at the origin of every fluid atom representing the contribution of the atom to the field quantity at $\boldsymbol{r}$. For the interpolation of different field quantities, different Smoothing Kernels can be chosen, but in general it is desired to have a function that is strongest around the origin and diminishes when moving to the outside, because the influence of an atom will diminish when moving away from its centre.

The following example shows one of the possible Smoothing Kernel functions:

$$W_{spiky}(\boldsymbol{r},h) = \frac{15}{\pi \cdot h^6} \begin{cases} (h - |\boldsymbol{r}|)^3 & 0 \le |\boldsymbol{r}| \le h \\ 0 & otherwise \end{cases}$$

Example 1

with a plot of the function values along $|\boldsymbol{r}|$ in [Figure 5].

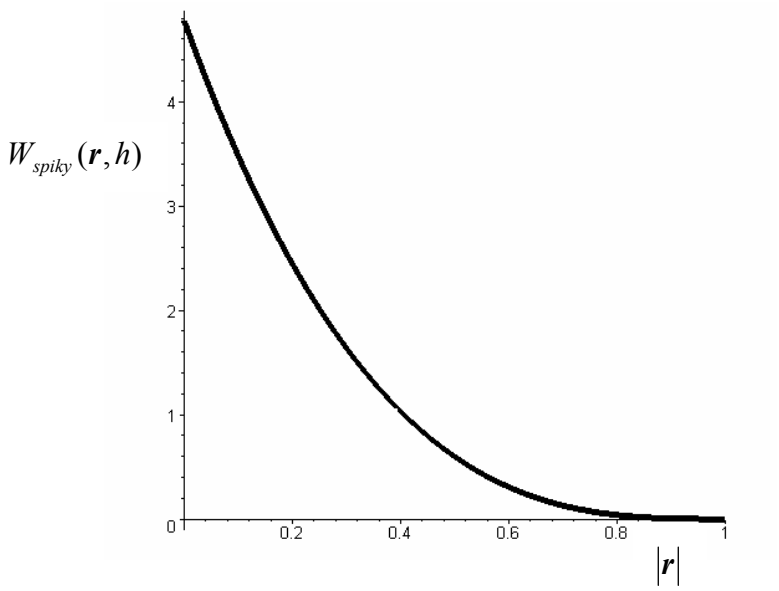


Figure 5

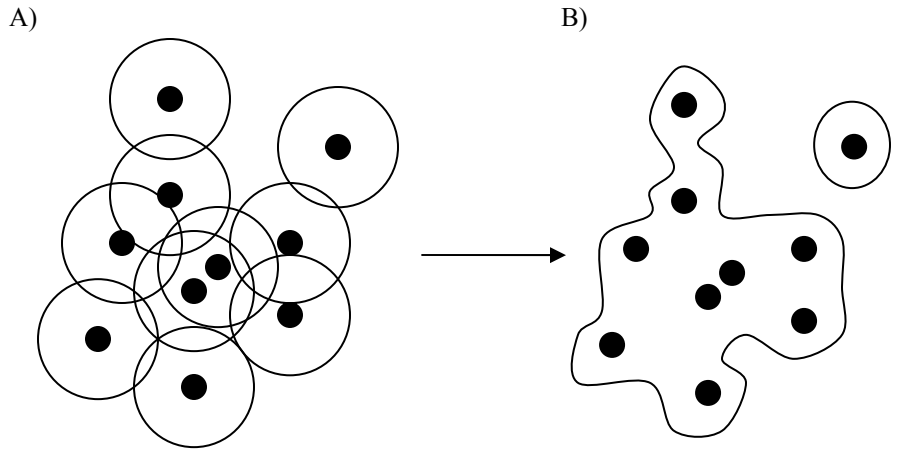
An example of a quantity that can be distributed by a smoothing kernel is the density of an atom. The distribution yields a density field, and is described according to the following formula:

$$\rho(\boldsymbol{r}) = \sum_j m_j \cdot W(\boldsymbol{r} - \boldsymbol{r}_j, h)$$

Example 2

where for every atom j, its mass m_j is distributed around its position r_j via the Smoothing Kernel W. The sum of these individual density distributions evaluated at a position $\boldsymbol{r}$ yields the final resulting density of the interpolation.

This distribution is visualised in [Figure 6].



A) Atom masses are distributed around their positions to form a density field. B) Defining field values above a certain threshold to be in the surface and the rest outside, one obtains a blobby surface

Figure 6

A Smoothing Kernel function has some special properties, namely that it is even

$$W(\boldsymbol{r},h)=W(-\boldsymbol{r},h)$$

Equation 8

is 0 outside its radius *h*

$$|\boldsymbol{r}|>h \Rightarrow W(\boldsymbol{r},h)=0$$

Equation 9

and normalized:

$$\int_r W(\boldsymbol{r},h)dr=1$$

Equation 10

Finally, the function has to be derivable over $|\boldsymbol{r}|$ twice, for an approximation of second order accuracy is desired. Note that while [Example 1] is derivable over $|\boldsymbol{r}|$, it is not derivable over $\boldsymbol{r}$, because the derivative of $W(\boldsymbol{r})$ over $\boldsymbol{r}$ is not continuous at $\boldsymbol{r}=\boldsymbol{0}$. In practice such a situation almost never occurs; to still be able to calculate the derivative of $W(\boldsymbol{r})$ over $\boldsymbol{r}$, one could define it to be 0 at $\boldsymbol{r}=\boldsymbol{0}$.

The general formula for interpolating scalar quantity A at location $\boldsymbol{r}$ by using $W(\boldsymbol{r},h)$ – as established in [I], [IX] and [X] – is very similar to [Example 2]. It consists of the following weighted sum of particle contributions, in [Equation 11]

$$A(\boldsymbol{r}) = \sum_j V_j \cdot A_j \cdot W(\boldsymbol{r} - \boldsymbol{r}_j, h) = \sum_j m_j \cdot \frac{A_j}{\rho_j} \cdot W(\boldsymbol{r} - \boldsymbol{r}_j, h)$$

Equation 11

where j iterates over all fluid atoms, m_j is the mass of atom j, V_j is the volume and ρ_j is the density at the atom position $\boldsymbol{r}_j$. The equation shows two factors for the contribution of a fluid atom to the final value, namely the Smoothing Kernel and the atom volume. Logically, an atom with a larger volume also has a larger contribution to the end result. The mass m_j is a fixed value for every fluid atom, but to establish the volume ρ_j is required as well. As already shown in [Example 2], this quantity is in itself a result of SPH interpolation, and its calculation will be explained fully in section [2.4].

Sometimes, not only the values of field quantities are required, but also their derivatives. Because the volume V_j and quantity value A_j belonging to fixed atom positions do not change for different values of $\boldsymbol{r}$, calculating the derivative of a quantity only involves taking the derivative of the Smoothing Kernel. Therefore, the gradient of quantity A is defined by:

$$\nabla A(\boldsymbol{r}) = \sum_j m_j \cdot \frac{A_j}{\rho_j} \cdot \nabla W(\boldsymbol{r} - \boldsymbol{r}_j, h)$$

Equation 12

and the same holds for the Laplacian or the second derivative of some interpolated quantity:

$$\nabla^2 A(\boldsymbol{r}) = \sum_j m_j \cdot \frac{A_j}{\rho_j} \cdot \nabla^2 W(\boldsymbol{r} - \boldsymbol{r}_j, h)$$

Equation 13

2.3.2 Interpolated Field Quantities

As mentioned, the fluid simulation technique in [I] is based on SPH, and therefore interpolates various field quantities using [Equation 11]. The first required field quantity to be interpolated is density ρ_j, which itself is a parameter of the interpolation equation. However, when substituting ρ_j into the equation [I] obtains:

$$\rho(\boldsymbol{r}) = \sum_j m_j \cdot \frac{\rho_j}{\rho_j} \cdot W(\boldsymbol{r} - \boldsymbol{r}_j, h) = \sum_j m_j \cdot W(\boldsymbol{r} - \boldsymbol{r}_j, h)$$

Equation 14

which is the equation from [Example 2]. This means that the density of the fluid at every atom position can be calculated now, in order to be used in [Equation 11] and [Equation 3].

The other quantity field of [Equation 3] that has to be interpolated in order to find the acceleration of a fluid atom is $\boldsymbol{f}$. As noted, f consists of three components: $\boldsymbol{f}_{pressure}$, $\boldsymbol{f}_{viscosity}$ and $\boldsymbol{f}_{external}$. The last component is determined by the environment of the fluid, while the first two components are (derivatives of) field quantities interpolated by SPH.

The first component, $\boldsymbol{f}_{pressure}$, is the gradient of the pressure field. According to the rule of interpolating gradients from [Equation 12], the following equation is derived:

$$\boldsymbol{f}_{pressure}(\boldsymbol{r}) = -\nabla p(\boldsymbol{r}) = -\sum_j m_j \cdot \frac{p_j}{\rho_j} \cdot \nabla W(\boldsymbol{r} - \boldsymbol{r}_j, h)$$

Equation 15

where the pressure p_j can be derived from the density ρ_j by using [Equation 4].

However, two interacting atoms i and j may not have the same position, so there might exist a difference in density and pressure for both atoms. In that case, calculation of the pressure force of atom i on atom j does not yield a symmetric result when compared to the force of atom j on atom i. Therefore, [I] proposes a fix to the above equation by taking the arithmetic mean of the pressures:

$$\boldsymbol{f}_{pressure}(\boldsymbol{r}) = -\sum_j m_j \cdot \frac{p + p_j}{2\rho_j} \cdot \nabla W(\boldsymbol{r} - \boldsymbol{r}_j, h)$$

Equation 16

where p is the pressure at $\boldsymbol{r}$.

The second component, $\boldsymbol{f}_{viscosity}$, can be determined in the same manner as $\boldsymbol{f}_{pressure}$. However, in this case [Equation 13] has to be used, because the viscosity is calculated by taking the Laplacian of the velocity field. Applying the SPH rule from [Equation 11] to the viscosity term again yields asymmetric forces between an atom i and j, as was the case with [Equation 15]. A fix is applied by [I] replacing the absolute velocity term $\boldsymbol{v}_j$ for every atom j with the relative velocity ($\boldsymbol{v}_j$-$\boldsymbol{v}$), yielding the following equation:

$$\boldsymbol{f}_{viscosity}(\boldsymbol{r}) = \mu \nabla^2 \boldsymbol{v} = \mu \sum_j m_j \cdot \frac{\boldsymbol{v}_j - \boldsymbol{v}}{\rho_j} \cdot \nabla^2 W(\boldsymbol{r} - \boldsymbol{r}_j, h)$$

Equation 17

where $\boldsymbol{v}$ is the velocity at $\boldsymbol{r}$.

The density and force density field can be calculated with the [Equation 14], [Equation 16] and [Equation 17], which means the acceleration of every fluid atom is derivable from this point forward as well.

Using the density values, another important concept of the fluid simulation can be determined: the fluid surface. While this has no influence on the calculations of the simulation itself, it will be used during visualisation. In order to be certain that the fluid surface can be recreated later on, the derivation of it is treated at this point already. This will make explicit which parameters will be required for the simulation output.

To calculate the surface of the fluid, [I] constructs a "color field", which assigns a value of 1 to the exact locations of a fluid atom and 0 to all other positions. The SPH method then distributes these values in space, creating a smoothed scalar field from which an isosurface can be taken for any arbitrary positive value. The isosurface is the surface of the fluid itself. The color field function *c* is as follows:

$$c(\boldsymbol{r}) = \sum_j m_j \cdot \frac{1}{\rho_j} \cdot W(\boldsymbol{r} - \boldsymbol{r}_j, h)$$

Equation 18

Note that in fluid dynamics, $\frac{m}{\rho} = V$, essentially indicating that [Equation 18] is an interpolation of the fluid volume, thereby a suitable choice for defining the volume boundary.

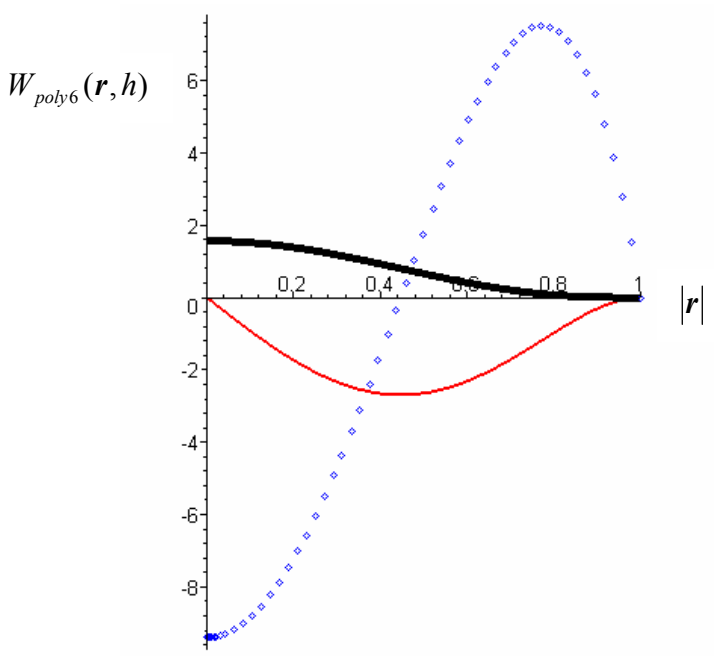
2.3.3 Matching Smoothing Kernels to Field Quantities

Now that the concept of Smoothing Kernels is clear, and the basic equations of a fluid simulation have been established by using SPH, the task that remains is to find a suitable Smoothing Kernel for every field quantity. More specifically, the choice has to be such that the distribution of values of the quantity around the origin of the Smoothing Kernel is as realistic as possible.

[I] already covers options for the various field quantities, but since one is free to choose their own kernels, it is good to briefly outline the decisions again. For the density field, [I] uses the W_{poly6} kernel with vanishing gradient near the origin:

$$W_{poly6}(\boldsymbol{r}, h) = \frac{315}{64 \cdot \pi \cdot h^9} \begin{cases} (h^2 - |\boldsymbol{r}|^2)^3 & 0 \le |\boldsymbol{r}| \le h \\ 0 & otherwise \end{cases}$$

Equation 19



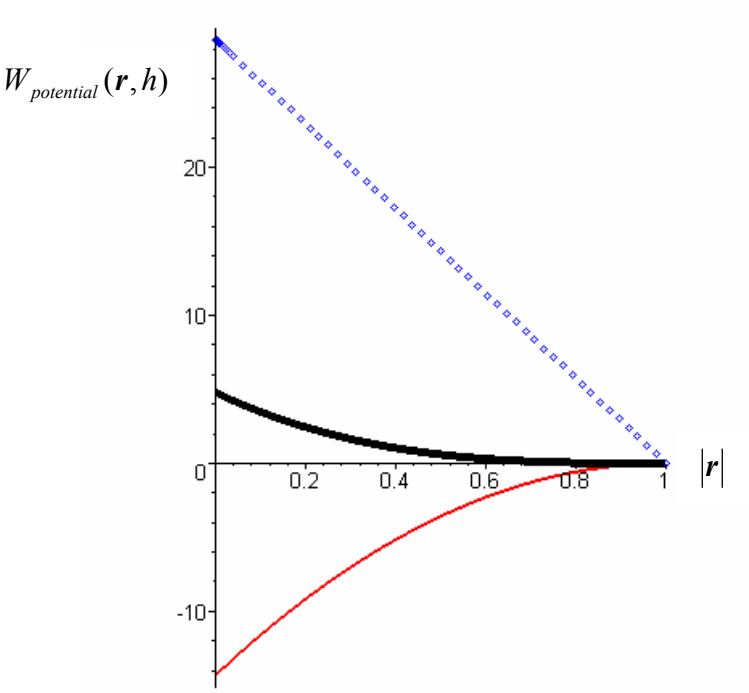
Plot of the W_{poly6} kernel. The thick black line is the original function, the thin red line the first derivative and the blue dotted line the second derivative.

Figure 7

Unlike the density field, the $\boldsymbol{f}_{pressure}$ field is based on the gradient of a Smoothing Kernel. It is desired to have a gradient that is non-vanishing near the origin, because the pressure has to become stronger when the distance between atoms decreases. The kernel $W_{potential}$ is a good choice for the $\boldsymbol{f}_{pressure}$ field, since it meets these requirements:

$$W_{potential}(\boldsymbol{r},h) = \frac{15}{\pi \cdot h^3} \begin{cases} (1-|\boldsymbol{r}|/h)^3 & 0 \le |\boldsymbol{r}| \le h \\ 0 & otherwise \end{cases}$$

Equation 20



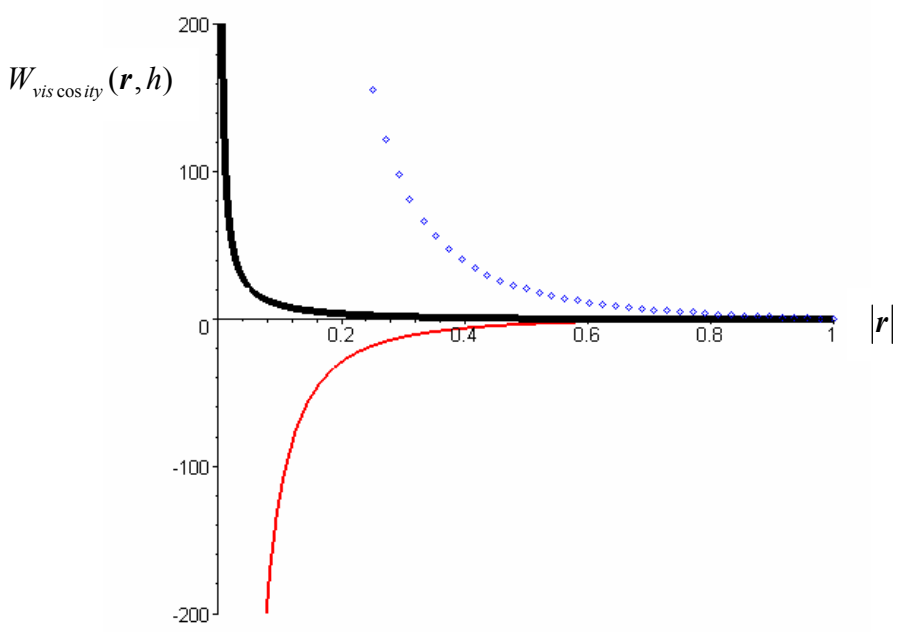
Plot of the $W_{potential}$ kernel. The thick black line is the original function, the thin red line the first derivative and the blue dotted line the second derivative.

Figure 8

For the computation of $\boldsymbol{f}_{viscosity}$, the $W_{viscosity}$ kernel of [I] is a good choice, since the viscosity calculations involve the Laplacian of the kernel, which has the undesired property of being negative for a large part of the interval of $W_{potential}$ and W_{poly6}.

$$W_{viscosity}(\boldsymbol{r},h)=\frac{15}{2\cdot\pi\cdot h^3}\begin{cases}(-\frac{\boldsymbol{r}^3}{2h^3}+\frac{\boldsymbol{r}^2}{h^2}+\frac{h}{2\boldsymbol{r}}-1) & 0\le|\boldsymbol{r}|\le h\\ 0 & otherwise\end{cases}$$

Equation 21



Plot of the $W_{viscosity}$ kernel. The thick black line is the original function, the thin red line the first derivative and the blue dotted line the second derivative.

Figure 9

Other quantity fields, like the (gradient field of a) color field used for fluid surface and surface tension computations in [I], can use the W_{poly6} kernel.

2.4 Solving the Model

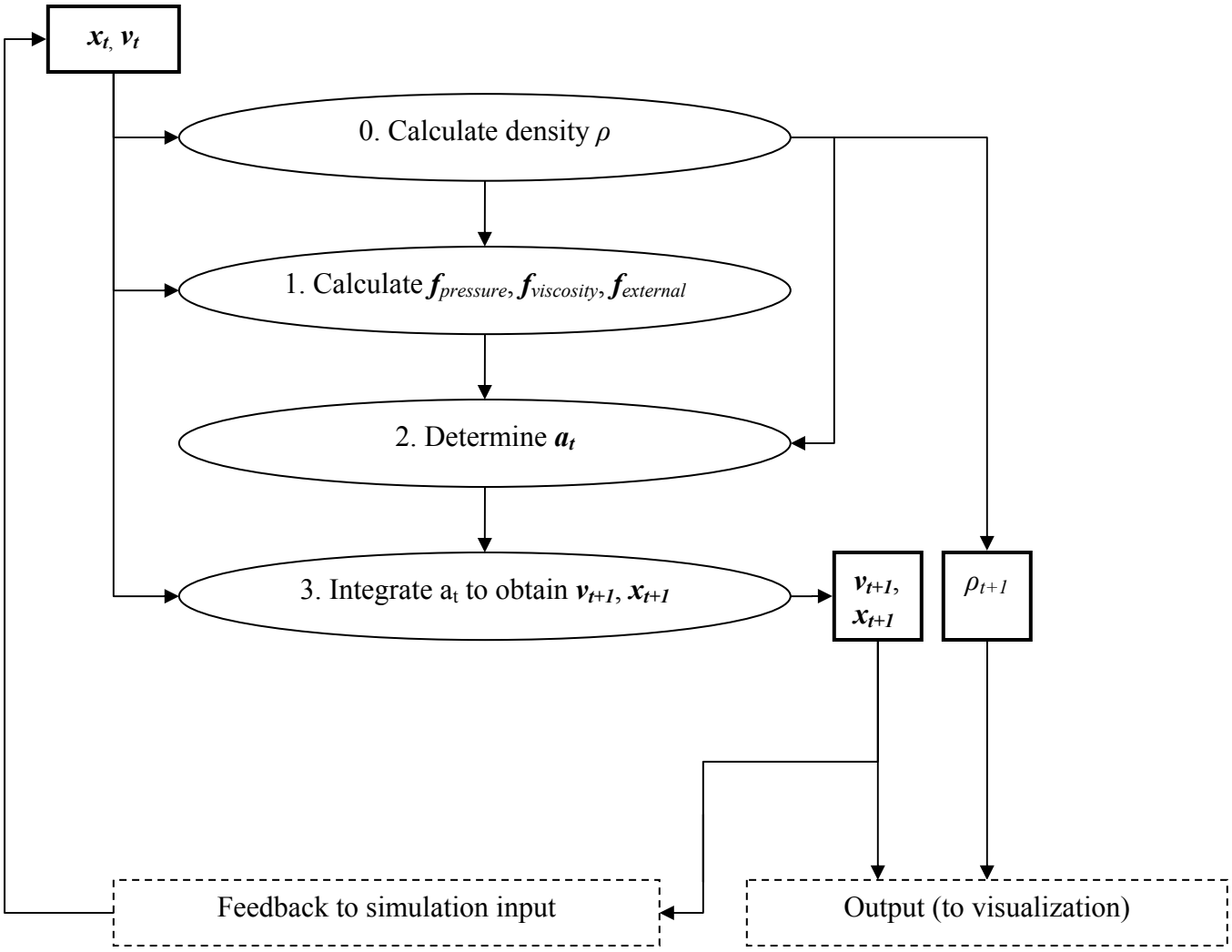
2.4.1 System of Equations

The goal of the following section is to present a computational way to solve the system of equations that relate our field quantities. Furthermore, it is necessary to determine the simulation quantities we want to visualise, as well as determine how the simulation feeds back on itself.

At the start of the simulation, every particle has an initial position and velocity. The job of the simulation is now to calculate the new accelerations and integrate these to obtain the velocities and positions of the next point in time.

According to [Equation 3], calculating the accelerations depends on $\boldsymbol{f}_{pressure}$, $\boldsymbol{f}_{viscosity}$, $\boldsymbol{f}_{external}$ and a density at the position of each fluid atom. These attributes are in turn defined by [Equation 14], [Equation 16] and [Equation 17]. Upon closer inspection, the latter two equations require the density at atom positions as well, together with the atom velocities.

This implies an order of calculation for every atom j displayed by [Figure 10].



The fluid atom simulation, with square boxes denoting the input and output attributes of the simulation steps, ellipses denoting the simulation steps themselves and dashed boxes denoting what happens after a simulation iteration has finished. Explicit mention of the in- and output parameters between consecutive simulation steps has been omitted for readability; they are the result of the calculation of the previous simulation step.

Figure 10

First, the density is calculated (0). Hereafter, the forces on particles can be established via the densities, particle positions and particle velocities (1). Then, the acceleration per particle is calculated (2), after which the new particle velocities and positions are integrated (3) and sent to the output along with the previously calculated density at each position.

In [Figure 10] one can see that the simulation keeps updating the position and velocity of every fluid atom. As output to the visualisation, the density parameter is required as well. To see why, one should look at the color field function defining the fluid surface, from [Equation 18]. The equation dictates that not only the position of the fluid atoms has to be known, but also the fluid density at that position. The mass is assumed to be the same for

every particle. With this information the surface of the fluid can be determined for visualisation algorithms.

One is free to choose the method of integration for the last step of the simulation; simple Euler integration suffices:

$$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t + \boldsymbol{a}_t \cdot \Delta t$$

and

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \boldsymbol{v}_t \cdot \Delta t$$

Equation 22

where the subscripts denote successive moments in time and Δt the difference in time between these successive moments.

3 Fluid Visualisation with Surface Particles

3.1 Introduction

While section [1.2] shows that the visualisation of a fluid surface gives rise to many possible solutions, this work focuses on a specific technique: visualisation with surface particles. That is, the isosurface of the volume quantity – computed by interpolation in [Equation 18] – is described by a number of surface particles. In contrast to the fluid atoms which are used to compute the field quantities describing the flow, surface particles are constrained to stay on the isosurface. This constitutes the first part of the visualisation algorithm. However, constraining particles to the surface of the fluid is not sufficient. Some particles could overlap, essentially rendering the same part of the surface twice, while some parts of the surface may not be rendered at all. In the first case, the performance of the rendering is negatively affected, while the second case is detrimental to the visual quality; holes may form in the fluid surface. Therefore, particles need to distribute themselves evenly across the fluid surface. This is the second part of the visualisation algorithm. The distribution of particles can in itself be divided into two parts as well: the first part deals with local distribution of particles within the small areas around them, and the second part deals with global distribution of particles on different – maybe even disconnected – parts of the fluid surface.

First, the problem statements for both parts are formalised in [3.2]. Then, [3.3] chooses a model formulating a solution to the problem statements. Succeeding the explanation of the models are the methods in section [3.4], which interpolate the newly introduced field quantities. The whole overview of the system of equations is given in [3.5]. Section [3.6] then presents the implementation details required to construct the algorithms mapped onto the corresponding hardware. Consequently, a test environment is set up in [3.7]. Using a test program, multiple parameters of the models will be varied in order to investigate where the bottlenecks in terms of performance of the algorithms are located, and to find out when the quality of rendering is at its best. Section [3.8] considers possibilities to expand upon the theories discussed up to that point.

For every one of the aforementioned sections, the first part of the visualisation is discussed first, and the second part of the visualisation is discussed last. This causes the explanation of the first part to be distributed among sections [3.2.1], [3.3.1] and [3.4.1]. For the second part, the explanation is given in sections [3.2.2], [3.3.2], [3.3.3] and [3.4.2]. Implementation details follow in sections [3.6.2] and [3.6.3] for the first part, with section [3.6.4] aimed at the second part. Finally, the test program will test the performance of the first part in [3.7.2], with the visual appearance of the second part in [3.7.3] and [3.7.4].

3.2 Problem Statement

3.2.1 Constraining the Particles

Given the function F:

$$F(\boldsymbol{x}) \quad :\Re^3 \rightarrow \Re$$

Equation 23

visualise the implicit function generated by taking the isosurface of F at a certain threshold T:

$$F(\boldsymbol{x}) = T$$

Equation 24

in an efficient and as accurate as possible manner.

The need for efficiency is already reflected in the choice for a particle-based simulation. The order of accuracy will depend on the choice of particle parameters, namely their amount, size and look.

The notion of having to visualise an implicit function immediately raises the question as to what kind of function this will be. As explained section [2] describing the fluid simulation, the objective is to visualise the color field of [Equation 18] generated by fluid atoms moving in space. This color field requires as input the position of the atom and the density at the position of the atom. Therefore function F is changed to become a function of the evaluation position and the required input-vector $\boldsymbol{q}$ of atom positions and densities:

$$F(\boldsymbol{x}, \boldsymbol{q}(t)) \quad :\Re^3 x \Re^m \rightarrow \Re$$

Equation 25

where m is the number of atoms. Time-varying vector $\boldsymbol{q}$ should be the concatenation of all fluid atom positions with the atom densities, because these properties define the fluid surface of [Equation 18]. Formally:

$$\boldsymbol{q} = (\boldsymbol{a}_1, ..., \boldsymbol{a}_m, \boldsymbol{\rho}_1, ..., \boldsymbol{\rho}_m)$$

Equation 26

where $\boldsymbol{a}_j$ is the position of atom $0<j\leq m$, and ρ_j its density.

As we're dealing with particles, the first part of the problem statement can finally be summarised by a constraint placed on the motion of all particles $1..n$, such that for implicit function F the following holds at every point in time:

$$F(\boldsymbol{p}_i(t), \boldsymbol{q}(t)) = T$$

Equation 27

where $\boldsymbol{p}_i$ is the position of every particle $0<i\leq n$, F is the color field of [Equation 18] and with time-varying vector $\boldsymbol{q}$ defining the fluid surface. Simply put, all the particles should be on the surface of the fluid defined by the fluid atoms.

3.2.2 Distributing the Particles

The second problem involves the distribution of particles over the fluid surface. Say the density of particles on the fluid surface is denoted by the following function

$$\sigma(\boldsymbol{r}) \quad :\Re^3 \rightarrow \Re$$

Equation 28

for a position $\boldsymbol{r}$. This density can be expressed as the number of particles per unit surface of the fluid surface for positions on this surface, which is a function in world-space coordinates.

However, one could also model σ as a function in screen-space coordinates, relating a screen-space two-dimensional point to which the fluid surface maps, to a screen-space fluid density:

$$\sigma(\boldsymbol{r}) \quad :\Re^2 \rightarrow \Re$$

Equation 29

in other words, this function outputs the number of particles per unit surface of screen space.

In any case, the definition of surface particle density is:

$$\sigma(\boldsymbol{r}) = \frac{\sum_{i|\, p_i \in A} m_i}{A}$$

Equation 30

where m_i is the mass of particle i, $\boldsymbol{p}_i$ is the position of particle i, and A the surface area over which the density is calculated.

An even distribution is defined as the situation in which all densities at the position of every particle on the fluid surface are equal. Formally, the goal is to minimize the following function:

$$E(\sigma(\boldsymbol{p}_1),\ldots,\sigma(\boldsymbol{p}_n)) = \sum_i \sum_j \left(\sigma(\boldsymbol{p}_i) - \sigma(\boldsymbol{p}_j)\right)^2$$

Equation 31

where i and j range over all particles [1..n] on the fluid surface, and $\boldsymbol{p}_i$ is the position of particle i.

In a purely mathematical sense [Equation 31] could have many solutions; as long as all particle positions yield the same density it does not matter what the actual value is. In the case of a fluid surface however, there is only one solution to [Equation 31], for the surface is finite in size and the number of particles is fixed. The solution to the equation will always be the number of particles divided by the area of the surface.

3.3 Abstract Model

3.3.1 Equations of Motion for Constrained Surface Particles

Constraining particles to an implicit surface is a solved problem. The solution to this problem is outlined in [II]. Just like the fluid simulation, the visualisation algorithm will essentially be a simulation of the movement of particles. However, while the fluid simulation calculates an acceleration based on realistic fluid forces, the visualisation does not require natural movement of the surface particles. Instead, it calculates velocities to constrain particles onto the surface of the fluid and integrates the new particle positions based on these constrained velocities.

The article presents two constraint equations, one to move the particles towards the surface, and one to move the surface towards the particles. In the case of SPH fluid simulation, the fluid atoms with their surface move independent of the location of the surface particles, so the second equation can be omitted.

An overview of the derivation performed in [II] is given below, with the final equation for constraining the particle velocities being presented by [Equation 36].

To satisfy [Equation 27], it is enough to satisfy it at $t = 0$ and make sure that

$$\dot{F}_i = F_i^x \bullet \dot{\boldsymbol{p}}_i + F_i^q \bullet \dot{\boldsymbol{q}} = 0$$

Equation 32

For every iteration, where $\dot{p}_i$ now denotes the velocity of particle i, F_i^x and F_q^i the derivatives of F_i over x and q at position $\boldsymbol{p}_i$, and $\dot{F}^i$ the derivative of F_i over time at position $\boldsymbol{p}_i$. The symbol $\bullet$ is the vector dot product.

[Equation 32] in itself is not enough, since drift may occur over time. [II] proposes a simple feedback term to correct the drift:

$$\dot{F} = -\phi F$$

Equation 33

where ϕ is the feedback constant. The complete constraint equation now becomes:

$$C_i\left(\boldsymbol{p}_i, \dot{\boldsymbol{p}}_i, \boldsymbol{q}, \dot{\boldsymbol{q}}\right) = F_i^x \bullet \dot{\boldsymbol{p}}_i + F_i^q \bullet \dot{\boldsymbol{q}} + \phi F_i = 0$$

Equation 34

To change a desired velocity $\boldsymbol{P}_i$ of particle i to satisfy the constraint C_i with a minimum impulse size, [II] shows that this constraining impulse – also called the objective function – should be a linear combination of the gradients of the constraint functions:

$$\boldsymbol{P}_i - \dot{\boldsymbol{p}}_i = \lambda_i F_i^x$$

Equation 35

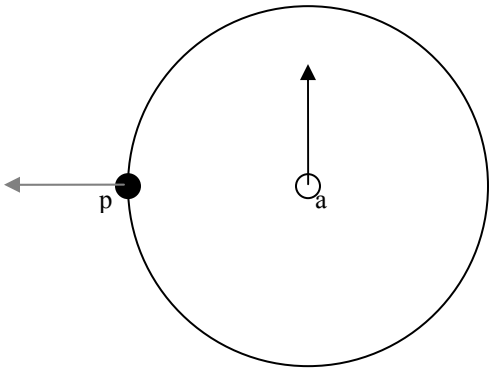
[II] shows that the unknown λ_i can be replaced by substitution of [Equation 35] for $\dot{p}_i$ in [Equation 34], and solving for λ_i. This will result in the final equation for $\dot{p}_i$:

$$\dot{p}_i = \boldsymbol{P}_i - \frac{F_i^x \bullet \boldsymbol{P}_i + F_i^q \bullet \dot{\boldsymbol{q}} + \phi F_i}{F_i^x \bullet F_i^x} F_i^x$$

Equation 36

No matter what the choice is for desired velocity $\boldsymbol{P}_i$, it is always possible to constrain the velocity according to [Equation 36], yielding a final particle velocity and integrate this to obtain the particle position.

The question remains as to what the right choice for $\boldsymbol{P}_i$ should be. A possible observation could be that as long as particles are on the fluid surface, they should not move. A naive solution is to assign a zero velocity to $\boldsymbol{P}_i$. However, this introduces the problem illustrated by [Figure 11], in which fluid atom moves in a direction orthogonal to F^q.

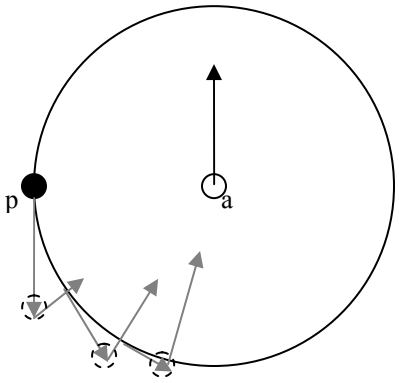


A situation consisting of one surface particle p, and one fluid atom a. The fluid surface is visualised by the circle around a. The surface particle is located on the fluid surface. The black arrow denotes the direction in which the fluid atom moves, while the grey arrow visualises F^q at p.

Figure 11

The result of this situation, when combined with the choice for a zero desired velocity $\boldsymbol{P}_i$, is that the surface particle will not move. The solution to [Equation 36] yields a zero-length constrained velocity. This result is quickly established by noting that the dot product with $\boldsymbol{P}_i$ is zero, the particle is located on the fluid surface already and the fluid atom velocity is orthogonal to F^q with no change in density over time at the fluid atom position, which equates to zero for the terms $F_i^x \bullet \boldsymbol{P}_i$, ϕF_i and $F_i^q \bullet \dot{\boldsymbol{q}}$ respectively.

The effect of the particle in [Figure 11] having a zero constrained velocity is that it will not move along with the fluid atom. It will lag behind and eventually end up at a point on the line going through the position of the fluid atom in the direction of its velocity, as demonstrated in [Figure 12].



The same particle and fluid atom as in [Figure 11], now visualised over time. The black dot is the particle at t_0, while the dashed dots represent the particle at t_1, t_2 and t_3. The black arrow denotes the fluid atom velocity, while the grey arrows denote the (negative) unconstrained and constrained component of the atom velocity one after another, for every new particle position. The unconstrained component causes a gap to originate between the surface and the particle, while the constrained component tries to accommodate for this along the gradient of the surface. During the particle's movement downward, the unconstrained component of the velocity becomes smaller, moving the particle closer to the surface. When the particle is finally on the fluid surface, it will have ended up below the fluid atom.

Figure 12

The conclusion which can be drawn from this example is that the desired velocity of the particle should be zero with respect to the velocity of the fluid atom, and not with respect to the absolute coordinate frame. In more general terms, the desired velocity of a surface particle is equal to the average velocity of its environment consisting of fluid atoms influencing the particle. Therefore, the desired velocity of a surface particle is taken to be a weighted average of the velocities of all fluid atoms defining the surface at its position:

$$\boldsymbol{P}_i^{env} = \frac{\sum_{j=0}^{m} w_j^i \dot{\boldsymbol{a}}_j}{\sum_{j=0}^{m} w_j^i}$$

Equation 37

with atom positions $\boldsymbol{a}_j$ and weighting parameters w_j^i of atom j at the position of particle i.

Instead of the regular atom velocities, it is also possible to choose the atom velocities after they are constrained to the fluid surface at the position of particle i, thereby effectively calculating the "constrained environment velocity".

The only question left then, is the choice for the weighting parameters in [Equation 39]. The observation is made that for a fluid atom j, the change in the density field over time

caused by j at position $\boldsymbol{p}_i$ of a certain particle i is not only proportional to j's velocity, but also to the length of the derivative of F over $\boldsymbol{a}_j$, evaluated at $\boldsymbol{p}_i$. This holds because the derivative equals the change in the fluid density with respect to every unit of movement along the principal axes. Therefore, the weight an atom j at position $\boldsymbol{p}_i$ is chosen to be proportional to the length of $F^{aj}(\boldsymbol{p}_i)$, since F^{aj} defines the rate of change in density caused by moving the fluid atom according to its velocity.

$$w_j^i = \begin{cases} \left|F_i^{aj}\right| & if \left|\boldsymbol{a}_j - \boldsymbol{p}_i\right| < h \\ 0 & otherwise \end{cases}$$

Equation 38

with h the Smoothing Kernel radius from the color field [Equation 18].

This concludes the analysis of the equations of motion for the surface particles, in order to constrain them to the fluid surface.

3.3.2 Local Particle Distribution by Repulsion

As can be seen from [Equation 31], the second goal of the visualisation is to minimize differences between densities at the positions of the surface particles. To this end, a local distribution algorithm is constructed in this section, while a global distribution algorithm can be found in [3.3.3].

In the local distribution algorithm, a repulsion force is introduced which moves particles in the direction of the gradient of their density field, repelling neighbouring particles as much as possible.

$$\boldsymbol{f}_{repulsion}(\boldsymbol{p}_i) = -\nabla\sigma(\boldsymbol{p}_i)$$

Equation 39

The repulsion force will always move a particle towards an area of lower density. To understand this, one should again realise that density differences need to be minimised. Moving a particle towards the area A of lower density increases the density of that area, while moving it away from area B of higher density decreases the density of area B. This minimises the density differences between area A and B.

To be able to use this repulsion force while still obeying to the constrained velocity of [Equation 36], the force is regarded as acceleration of the surface particles. This acceleration is integrated over time to form a desired repulsion velocity, defined by the following equation:

$$\boldsymbol{P}_i^{repulsion} = \int_t \left(\boldsymbol{f}_{repulsion}(\boldsymbol{p}_i)\right) dt$$

Equation 40

If so desired, this desired velocity can be dampened over time in order to slow particles down which aren't repelled anymore.

This desired velocity can then be plugged into [Equation 36] via the term $\boldsymbol{P}_i$, yielding the following expression:

$$\boldsymbol{P}_i = \boldsymbol{P}_i^{env} + \boldsymbol{P}_i^{repulsion}$$

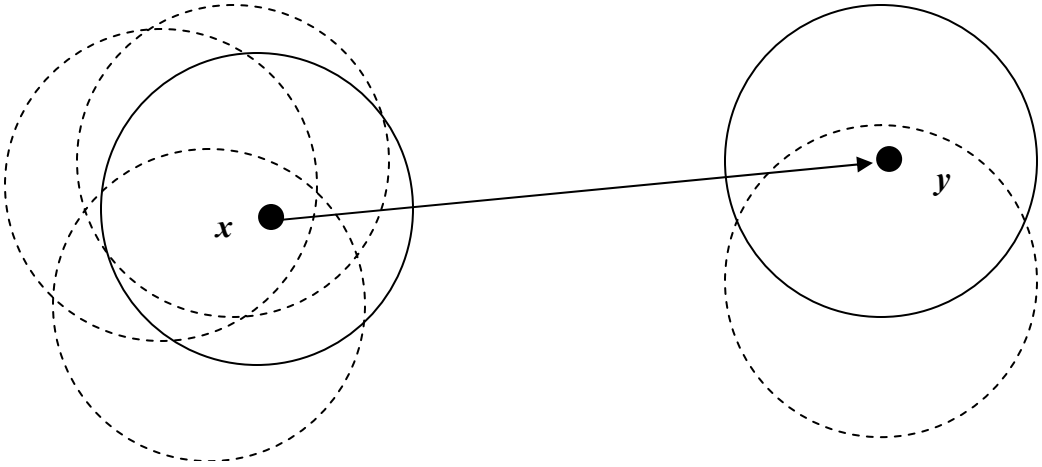
Equation 41

3.3.3 Global Particle Distribution by Dispersion

The local particle repulsion algorithm from section [3.3.2] focused on obtaining equal distributions within the immediate environment of a particle. The global repulsion algorithm however, focuses on density differences at positions with arbitrary distances. These positions could be near each other on the same surface, but they could just as well be located on different disconnected surfaces. The goal of the global method is to obtain an even global distribution; every (disconnected) part of fluid surface should have the same particle density as quickly as possible. This will be accomplished by removing particles from locations where the density is high, and redistributing them at locations where the density is low.

The global distribution system has to overcome one obstacle: find out which particles reside in "dense" areas and which particles reside in "sparse" areas. Half of the problem is easily solved; the density at arbitrary positions already has to be calculated to find the repulsion force in [Equation 39].

The problematic part of the aforementioned obstacle is to determine how dense or sparse a single calculated density is with respect to every area of all fluid surfaces. Theoretically, this would require an evaluation of all densities of every other particle, which is not an efficient solution. Therefore, the choice has been made for a stochastic model. In this model every surface particle chooses a random comparison particle once every t seconds. At the occasion of choosing the comparison particle, the densities of the two particles are compared and the original particle can determine if it wants to stay at its position or adopt the position of its comparison particle. A lower density at the position of the comparison particle might cause the original particle to change its position, to increase density at the comparison position and decrease it at the original position. Otherwise, the original particle will not move. This method is illustrated by [Figure 13].



The original surface particle at position ***x*** is moved to the comparison particle at position ***y***, because its density is higher than the density at the position of the comparison particle. This is indicated by the (dashed) spheres representing the influence radii of (neighbouring) particles.

Figure 13

So every t seconds, the number of comparison pairs is equal to the number of particles, which implies that every particle is featured in an expected number of two comparisons. Such a particle fulfils the role of "original particle" exactly once, and therefore it will be the "comparison particle" in the other comparison pair. In any of these two comparisons, the particle can either have the lower or the higher density. This generates four possibilities of equal probability for the outcome of the two comparisons featuring one common particle – this particle can have a lower or higher density in any of the two comparisons. This implies an expected number of one position-change per two comparisons, which can be checked by writing down the four different cases and determining the number of position-changes per case. In conclusion, the expected number of position-changes every t seconds equals the number of particles.

Changing the position of a particle on the basis of having a lower or higher density will create a very restless surface, as one might suggest from the evaluation above. A surface will never be of exactly equal density, and worse yet, the position change might increase the density difference function of [Equation 31]. This error function is desired to be as small as possible, but it is not possible to enforce such a constraint without evaluating the density at every particle-pair. Therefore, a different heuristic is constructed to determine whether to assign a new position to a particle. Now say the densities before a position change at position $\boldsymbol{x}$ and $\boldsymbol{y}$ are σ_{x0} and σ_{y0}, and after the position change σ_{x1} and σ_{y1}. The heuristic is as follows: only allow a change of a particle at $\boldsymbol{x}$ to position $\boldsymbol{y}$ if the difference $\Delta\sigma_1(\boldsymbol{x},\boldsymbol{y}) = (\sigma_{x1} - \sigma_{y1})^2$ in densities after the change is smaller than the difference $\Delta\sigma_0(\boldsymbol{x},\boldsymbol{y}) = (\sigma_{x0} - \sigma_{y0})^2$ in densities before the change. It is obvious that these are the error terms constituting [Equation 31].

So in short, to be able to change the position of a particle i from position $\boldsymbol{p_i}$ to position $\boldsymbol{p_j}$ from particle j, the following requirement should hold:

$$\Delta\sigma_1(\boldsymbol{x},\boldsymbol{y}) < \Delta\sigma_0(\boldsymbol{x},\boldsymbol{y})$$

Equation 42

where $\boldsymbol{x} = \boldsymbol{p}_i$ and $\boldsymbol{y} = \boldsymbol{p}_j$ before changing particle positions, $\Delta\sigma_0(\boldsymbol{x},\boldsymbol{y}) = (\sigma(\boldsymbol{x}) - \sigma(\boldsymbol{y}))^2$ before changing particle positions and $\Delta\sigma_1(\boldsymbol{x},\boldsymbol{y}) = (\sigma(\boldsymbol{x}) - \sigma(\boldsymbol{y}))^2$ after the changing particle positions.

3.4 Interpolation of Field Quantities

3.4.1 Particle Constraints

In essence, the discrete model of the implicit function F required for constraining the particles to the fluid surface has already been established by the fluid simulation. As noted in the problem statement, the color field from [Equation 18] is used as a function to generate the implicit surface with. The input to this equation consists of the particle positions and the densities at these points. This is reflected in the time-varying vector $\boldsymbol{q}$ from [Equation 26].

However, when looking at [Equation 36] that constrains the surface particles, one might notice that not only the function F with derivatives is a component of the velocity constraint equation, the derivative of vector $\boldsymbol{q}$ is part of it as well. This implies that the visualisation should also have knowledge about the derivative of its inputs – atom velocities and the fluid density derived over time at particle positions – which the fluid simulation has to supply.

Instead of complicating the already established interaction between surface simulation and visualisation, the implicit function used for visualising the fluid surface will be simplified. More precisely, the representation of the color field [Equation 18] in the fluid visualisation algorithm will be simplified to only be dependent on the fluid atom positions. To this end, the density term will be omitted from the color field in the visualisation algorithm from now on.

The implicit function for visualisation becomes:

$$F\left(\boldsymbol{x},\boldsymbol{q}(t)\right) = c\sum_j W\left(\boldsymbol{x} - \boldsymbol{a}_j, h\right) = T$$

Equation 43

ranging over all atoms $0<j\leq m$, with m the number of atoms and $\boldsymbol{a}_j$ the position of atom j, c a constant factor including the atom mass m_j, W a Smoothing Kernel, T a threshold value and with time-varying vector $\boldsymbol{q}$.

This also causes vector $\boldsymbol{q}$ to be simplified; densities are not required anymore, so it becomes the concatenation of fluid atom positions:

$$\boldsymbol{q} = (\boldsymbol{a}_1, ..., \boldsymbol{a}_m)$$

Equation 44

3.4.2 Particle Distribution

For particle distribution, both a local and global distribution algorithm has been constructed. While the local distribution algorithm evaluates the gradient of a density field to obtain a repulsion force by [Equation 39], the global algorithm evaluates the density field itself to determine differences at particle positions. It is not yet entirely clear how the density field and its gradient can be calculated in a discrete environment. This section will provide a solution to this problem.

Again, one can employ the concept of Smoothing Kernels in order to interpolate the density field, which is equal to calculating the density of the fluid itself with [Equation 14]:

$$\sigma(\boldsymbol{p}_i) = \sum_j m_j \cdot \frac{\sigma_j}{\sigma_j} \cdot W(\boldsymbol{p}_i - \boldsymbol{p}_j, h') = c \sum_j W(\boldsymbol{p}_i - \boldsymbol{p}_j, h')$$

Equation 45

with $\boldsymbol{p}_i$ the position of particle i, h' the radius of influence and c the constant mass or repulsion scaling factor for all particles. The gradient then becomes:

$$\boldsymbol{f}_{repulsion}(\boldsymbol{p}_i) = -\nabla \sigma(\boldsymbol{p}_i) = -c \sum_j \nabla W(\boldsymbol{p}_i - \boldsymbol{p}_j, h')$$

Equation 46

The Smoothing Kernel of [Equation 46] has only one requirement: its derivative should be non-vanishing.

Once the gradient of the density is known, the local distribution algorithm can calculate a repulsion force with [Equation 39]. The desired repulsion velocity is calculated according to [Equation 40]. The discrete version of this equation is defines as follows:

$$\boldsymbol{P}_i^{repulsion}(t_{new}) = \boldsymbol{P}_i^{repulsion}(t_{old}) + \boldsymbol{f}_{repulsion}(\boldsymbol{p}_i) \cdot \Delta t$$

Equation 47

where Δt is $(t_{new} - t_{old})$.

In the case of the global distribution algorithm, the condition of [Equation 42] should be satisfied. Using [Equation 45], the densities at particle positions can be calculated; it requires only the world positions of neighbouring particles. However, this only solves half the problem: [Equation 42] requires that the density difference $\Delta\sigma_1(\boldsymbol{x},\boldsymbol{y})$ between positions $\boldsymbol{x}$ and $\boldsymbol{y}$ after a particle position change is smaller than the difference $\Delta\sigma_0(\boldsymbol{x},\boldsymbol{y})$ before the position change. The density difference after the position change cannot directly be evaluated, and should therefore be derived. A certain bound will be constructed on $\Delta\sigma_0(\boldsymbol{x},\boldsymbol{y})$, determining when the position change will take place.

The contribution of a particle to the density field at its own position – the center of its Smoothing Kernel – equals a positive constant $k = cW(0,h')$, defined by [Equation 45]. The densities after the position change in terms of those before are: $\sigma_{x1} = \sigma_{x0} - k$ and $\sigma_{y1} = \sigma_{y0} + k$. So at a position change, $(\sigma_{x1} - \sigma_{y1}) = (\sigma_{x0} - \sigma_{y0}) - 2k$. To decrease $\Delta\sigma(\boldsymbol{x},\boldsymbol{y})$, one has to make sure that $\Delta\sigma_1(\boldsymbol{x},\boldsymbol{y}) < \Delta\sigma_0(\boldsymbol{x},\boldsymbol{y})$, according to [Equation 42]. In other words: $(\sigma_{x1} - \sigma_{y1})^2 < (\sigma_{x0} - \sigma_{y0})^2$. The solution to this equation is $|\sigma_{x1} - \sigma_{y1}| < (\sigma_{x0} - \sigma_{y0})$. To put it in terms of $\Delta\sigma_0(\boldsymbol{x},\boldsymbol{y})$: $|(\sigma_{x0} - \sigma_{y0}) - 2k| < (\sigma_{x0} - \sigma_{y0})$ should hold. This solves to $\sigma_{x0} - \sigma_{y0} > k$. As a result, two particles i and j should only change positions when the density difference at their positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ exceeds a certain threshold:

$$\sigma(\boldsymbol{p}_i) - \sigma(\boldsymbol{p}_j) > cW(0,h')$$

Equation 48

3.5 Solving the Model

3.5.1 Required Data for Transfer

To be able to calculate the solution to [Equation 36] for every particle i, a number of components need to be evaluated at particle position $\boldsymbol{p}_i$:

- F_i is the actual value of the implicit function at $\boldsymbol{p}_i$, which according to [Equation 14] requires the positions of all fluid atoms having an influence on the scalar field at position $\boldsymbol{p}_i$
- $\boldsymbol{P}^i$, the desired velocity, consists of $\boldsymbol{P}^{env}$ and $\boldsymbol{P}^{repulsion}$. $\boldsymbol{P}^{env}$ can be calculated according to [Equation 37] and [Equation 38] with particle velocities and positions, while $\boldsymbol{P}^{repulsion}$ can be calculated with [Equation 40] using particle positions only.
- $\dot{\boldsymbol{q}}$ is the concatenation of fluid atom velocities
- F_i^x and F_i^q

F_i^x defines the change of F over x at position $\boldsymbol{p}_i$, and can for a fixed $\boldsymbol{q}$ simply be regarded as the gradient of the implicit function F at $\boldsymbol{p}_i$. The evaluation of this function at $\boldsymbol{p}_i$ only requires the positions of all fluid atoms having an influence on the scalar field.

The calculation of F_i^q is almost identical to that of F_i^x. To this end, we divide vector F_i^q into its separate components:

$$F_i^q = \left(F_i^{a1},...,F_i^{an}\right)^T$$

Equation 49

where F_i^{aj} is the derivative of F_i over the position $\boldsymbol{a}_j$ of atom j. Because only the contribution of atom j is considered for a single component F_i^{aj}, one could just as well take [Equation 43], remove the contribution of all other fluid atoms, yielding the following function:

$$cW\left(\boldsymbol{x}-\boldsymbol{a}_j,h\right)$$

Example 3

and derive this term over $\boldsymbol{a}_j$.

Now, say F_i^{x-aj} is the component of F_i^x contributed by atom j. Looking back at [Equation 43], this contribution amounts to

$$cW\left(\boldsymbol{x}-\boldsymbol{a}_j,h\right)$$

Example 4

derived over x. Comparing [Example 3] with [Example 4], one will notice that they are the same. However, both expressions are derived over different arguments. The chain rule relates the two derivations:

$$\begin{aligned}
&\frac{\partial\left(cW\left(\boldsymbol{x}-\boldsymbol{a}_j\right)\right)}{\partial\boldsymbol{x}} \\
&= \frac{\partial\left(cW\left(\boldsymbol{x}-\boldsymbol{a}_j\right)\right)}{\partial\left(\boldsymbol{x}-\boldsymbol{a}_j\right)} \cdot \frac{\partial\left(\boldsymbol{x}-\boldsymbol{a}_j\right)}{\partial\boldsymbol{x}} \\
&= \frac{\partial\left(cW\left(\boldsymbol{x}-\boldsymbol{a}_j\right)\right)}{\partial\left(\boldsymbol{x}-\boldsymbol{a}_j\right)} \cdot -\frac{\partial\left(\boldsymbol{x}-\boldsymbol{a}_j\right)}{\partial\left(\boldsymbol{a}_j\right)} \\
&= -\frac{\partial\left(cW\left(\boldsymbol{x}-\boldsymbol{a}_j\right)\right)}{\partial\left(\boldsymbol{a}_j\right)}
\end{aligned}$$

Equation 50

The result of this derivation is that:

$$F_i^{aj} = -F_i^{x-aj}$$

meaning that deriving F over $\boldsymbol{q}$ can be expressed in terms of its derivation over $\boldsymbol{x}$.

The reason for [Equation 50] to hold can also be explained intuitively: an evaluation of F after changing the evaluation position x in the implicit function defined by a single atom is equivalent to an evaluation after moving the atom in the opposite direction while keeping the evaluation position fixed.

Using [Equation 50], F_i^q can be expressed as follows:

$$F_i^q = \left(F_i^{a1},...,F_i^{an}\right)^T = \left(-F_i^{x-a1},...,-F_i^{x-an}\right)^T$$

Equation 51

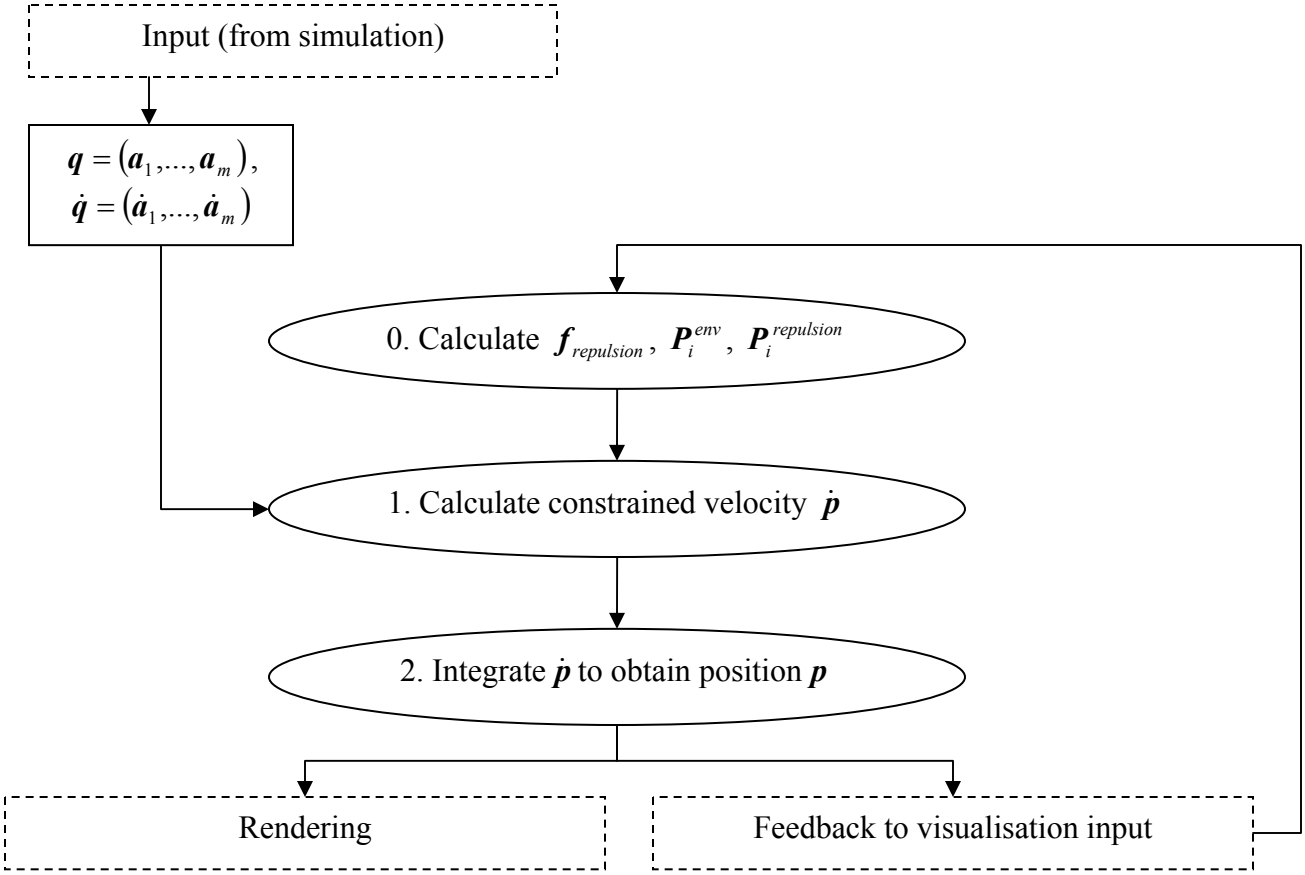
Because the calculation of F_i^q only requires components of F_i^x, the evaluation of the former function at $\boldsymbol{p}_i$ only requires knowledge about the positions of the fluid atoms.

In conclusion, it is correct to state that updating the velocity of a surface particle requires only the fluid atom positions and velocities.

3.5.2 System of Equations

Again, an overview of the visualisation model will be provided to aid in solving the different parts of the model and establish the order of calculation of the equations treated so far.

The overview is given by [Figure 14]. Based on the positions of the surface particles, the repulsion forces are calculated first, from which $\boldsymbol{P}^{repulsion}$ can be derived as well. On top of that, the particle velocities make it possible to determine $\boldsymbol{P}^{env}$ in step (0). The combination of $\boldsymbol{P}^{env}$ and $\boldsymbol{P}^{repulsion}$ enables the visualisation to determine the desired velocity $\boldsymbol{P}_i$. With $\boldsymbol{P}_i$ and the positions with velocities of the fluid atoms as input, [Equation 36] can be solved to obtain the constrained velocity for all particles in step (1). These velocities will be used as input to the position integration, which could be solved by any method, like Euler integration in step (2). Finally, the new positions of the particles are known and can be used for both rendering and the next visualisation iteration.

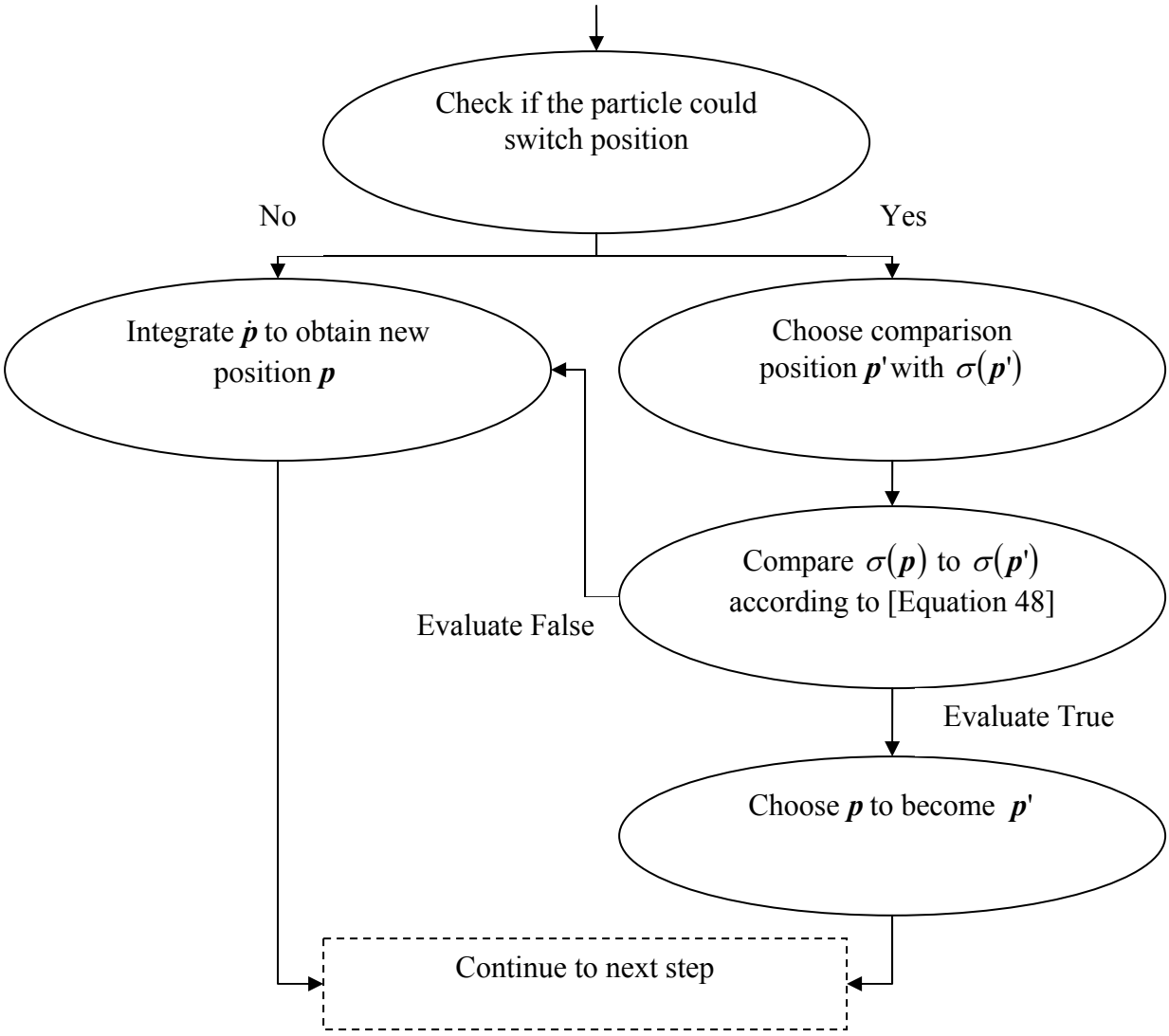


The fluid surface visualisation, with square boxes denoting the input and output attributes of the simulation and visualisation, ellipses denoting the visualisation steps themselves and dashed boxes denoting what happens after a visualisation iteration has finished. Explicit mention of the in- and output parameters between consecutive simulation steps has been omitted for readability; they are the result of the calculation of the previous visualisation step.

Figure 14

One element of the fluid simulation has not yet been incorporated into [Figure 14]. The figure lacks a global distribution algorithm, which removes particles from locations where the density is high, and redistributes them at locations where the density is low. This goal can be fulfilled at step (2), which is responsible for determining the positions of particles on the surface.

The full global distribution algorithm is summarised in [Figure 15], which is a replacement for step (2) in [Figure 14].



The replacement for the integration of surface particle positions in step (2) of [Figure 14]. This replacement adds the global distribution algorithm.

Figure 15

3.6 Implementation

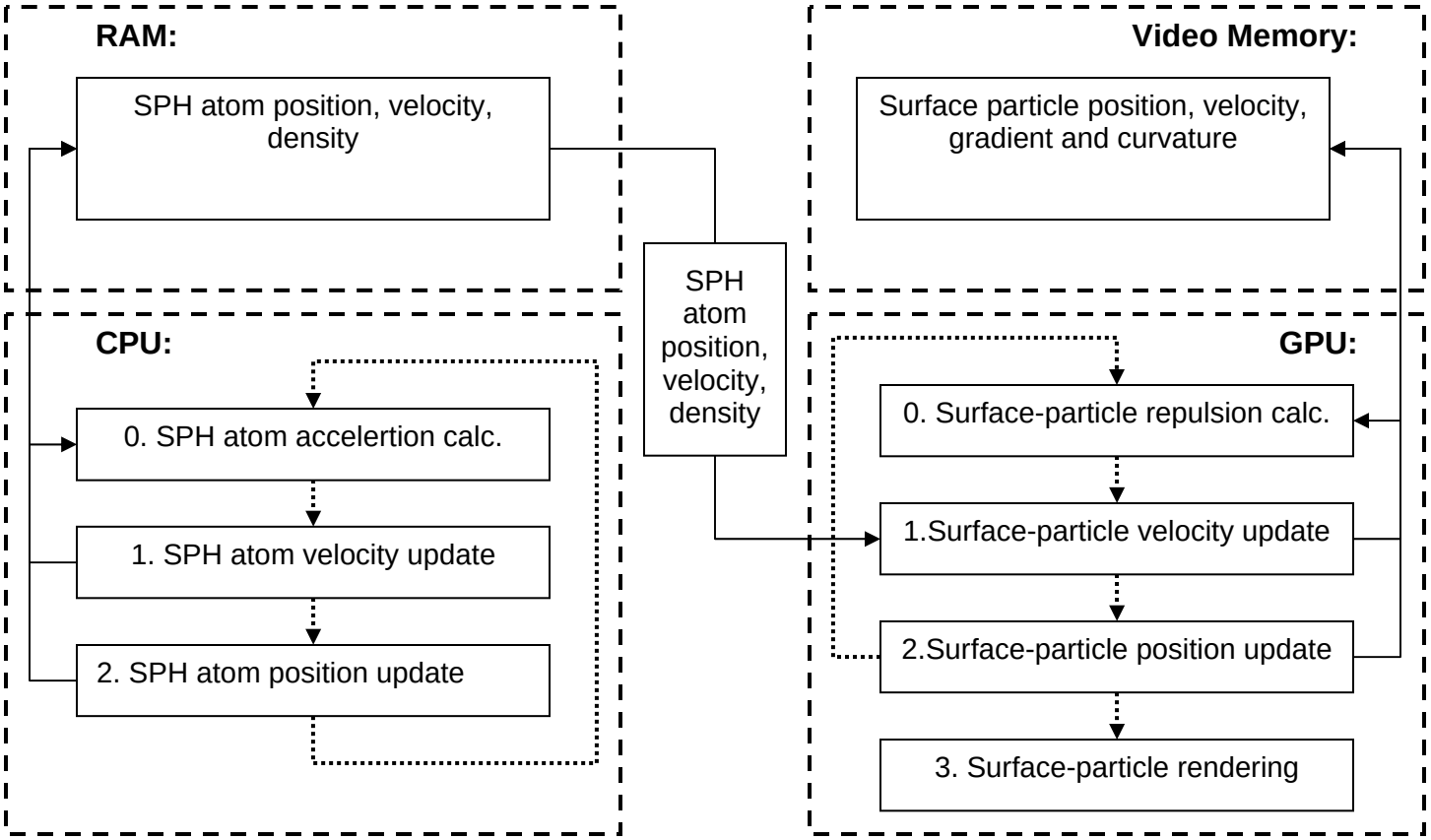
3.6.1 Mapping of models onto hardware

The visualisation algorithm will be implemented on top of the fluid simulation. Therefore, [Figure 16] shows a scheme with both the fluid atom simulation and fluid surface visualisation, with the transfer of data between them. Also, this scheme clarifies the mapping of the methods and their required data on the available processing units and memory.

To implement the previously discussed visualisation algorithm, [Figure 16] shows a surface particle simulation running on the GPU. Since it is already common for many simulations to be implemented on graphics hardware, the way in which forces, accelerations and velocities can be integrated to finally update particle positions is already known. The implementation of simulations like these is covered in [XVI], which explains how a general method of integration works on graphics hardware.

Still, the integration method consists of multiple steps, using constraint and repulsion forces to distribute particles on the fluid surface, relating to [Equation 43] and [Equation 46] respectively. To be able to calculate these forces, two problems need to be solved. To calculate the implicit function from [Equation 43], one could naively read all atom positions and construct the required sum of function values accordingly. However, this would be very inefficient on both the CPU and the GPU. Therefore all atoms have a radius – the same radius as the one used for the Smoothing Kernels – inside which they contribute to the scalar field. Outside of this radius, the atoms do not contribute to the scalar field. This means that with the right choice for the radius, only a small amount of atoms contribute to the value of a scalar field at a certain position. This can be exploited in a data structure, such that the GPU only needs a small number of atom positions to construct the value of a function for any location. So, the first problem to be solved is the manner in which to organize the data sent from the simulation on the CPU to the simulation on the GPU, such that the GPU simulation can construct the implicit function values as quickly as possible. This problem is treated in section [3.6.2], with implementation details in [3.6.3].

The same problem is applicable to the density field generated by the surface particles, required by [Equation 46] and [Equation 48] for particle distribution. It is very inefficient to have to reconstruct the density field value at a certain position by iterating over all particles, adding their contribution to the final value. Some kind of nearest-neighbour algorithm or data structure has to be constructed in this case as well, because the surface particles only influence a small portion of space. So the second problem concentrates on finding a method to calculate the density at every surface particle's position by evaluating only the particles in its neighbourhood. This problem is treated in section [3.6.4].



The fluid atom simulation running on the CPU/RAM, and the visualization running on the GPU/Video Memory. Black arrows indicate a read/write data dependency, while dotted lines indicate the order in which the simulations are being executed.

Figure 16

3.6.2 Choice of Data Structure for Simulation

First, the decision on the organisation of the fluid atom data sent from the simulation to the visualisation will be discussed.

As noted before, every fluid atom has a radius in which it influences the scalar field around its position. Outside this radius, the fluid atom does not interact with other fluid atoms or surface particles anymore. Therefore a particle is only influenced by a subset of all atoms and only needs to know the positions of the fluid atoms within the aforementioned radius of the particle's position. A good spatial data structure can help achieve this goal, minimising the amount of atom position lookups a surface particle has to perform. However, the construction of the structure has to be inexpensive, for it is handled on the CPU. Furthermore, the cost of a query in the data structure consists of navigating through the structure and reading the number of returned elements. Ideally they are both low, but for most data structures there is a trade-off between the two components.

The data structure of choice is a spatial hash, which has been studied in [III]. This particular structure maps 3D positions to a 1D hash index. This is accomplished by discretising 3D space into fixed axis-aligned bounding boxes, essentially creating a 3D grid with each box having the same size. The discretisation divides each component of the 3D position by the grid cell size, adds a constant and then rounds it down to the nearest integer. The following function illustrates this:

$$\boldsymbol{B}(x,y,z) = \left(\lfloor (x+c)/s \rfloor, \lfloor (y+c)/s \rfloor, \lfloor (z+c)/s \rfloor\right)$$

Equation 52

where $\boldsymbol{B}(x,y,z)$ is a 3-vector of integers identifying the box by a unique value, (x,y,z) the 3D point in space and s the cell size given to the boxes formed by the grid. The addition of the constant term c is used to eliminate symmetry around the origin.

Because we do not have unlimited space by which to identify every box in the grid along with its content, a number of buckets – the hash size – is chosen to hash our 3D grid cells into a 1D array of buckets. Later, the terms hash size and cell size will be used a lot more, so be aware that these terms are used for different concepts. While the hash size denotes the number of buckets in the hash, the cell size denotes the size of a 3D grid cell that is mapped to a bucket. The mapping should be as random as possible, because a certain spatial configuration should not cause a worse distribution of atoms than others. Ideally we want an arbitrary position to map uniformly at random to any of the buckets in the hash. We chose the following distribution function:

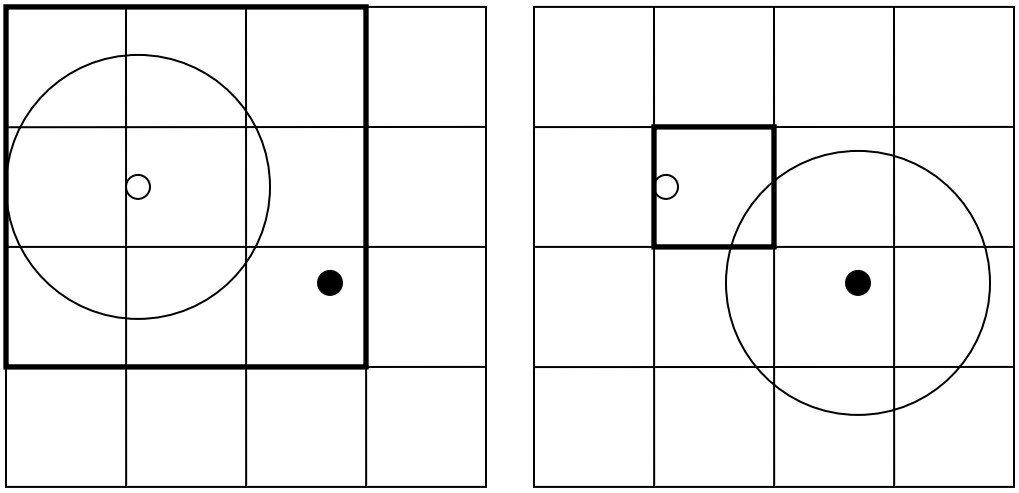
$$H(x,y,z) = \left(\boldsymbol{B}(x,y,z).x \cdot p1 + \boldsymbol{B}(x,y,z).y \cdot p2 + \boldsymbol{B}(x,y,z).z \cdot p3\right) \bmod h$$

Equation 53

where $H(x,y,z)$ maps 3-component vectors to an integer representing the index into the hash array of buckets, h is the hash size and $p1$, $p2$ and $p3$ are large primes. In our case, we chose 11113, 12979 and 13513, to make sure that the world size could still be reasonably large without exceeding the 23 bits available while calculating the hash function. Realise that a GPU has no integer arithmetic, so integers are replaced by floating point numbers with a 23-bit mantissa. This is a different hash function from the one described in [III], for the GPU does not have bitwise operators to replicate the hash function described in that paper.

Having established the hash function, there are two options when it comes to constructing the hash table and reading the data per particle in the surface visualisation. The first option, hash method A, adds every fluid atom to the hash table multiple times at construction, once for every grid cell it intersects or encapsulates. Consequently, the surface visualisation only has to query the hash table with the position of every surface particle once, for all the atoms intersecting or encapsulating the same grid cell have been added to the same bucket as well. The second option, hash method B, adds every fluid atom to the hash table only once, namely to the hash bucket that its position maps to. The fluid surface visualisation then reads the atom positions from every hash bucket corresponding to grid cells encapsulated or intersected by a sphere around the surface

particle with the radius of the fluid atoms. This way, every atom which possibly has an influence at the position of the surface particle will be evaluated. The difference is visualised in [Figure 17].



Hash method A is depicted on the left, and hash method B on the right. The white dot is a fluid atom, and the black dot a surface particle. Hash method A adds the fluid atom to every cell it influences, with the influence radius drawn by the surrounding circle and the filled grid cells with the thick black square. The particle now only has to query its own hash cell. Hash method B adds the fluid atom to just one grid cell, requiring the particle to query all cells in the radius of influence around it.

Figure 17

First, the query time per particle of the visualisation algorithm will be considered. For hash method A, the surface particle only has to query its position. Hash method B has to query hash buckets for every grid cell it encapsulates, but this is a constant q, for the radius never changes. Worst case, the querying therefore consists only of calculating the constant number of grid cells, $O(1)$, and reading the number of returned atom positions for the velocity equation $O(k)$, for k returned atom positions.

Querying the hash on the GPU is of complexity O(k)

Theorem 1

Constructing the hash table is most expensive for hash method A: for every atom, all grid cells (partially) within its volume are scanned, after which the atom is added to the hash buckets corresponding to these grid cells. The cost for doing this is $q*O(n_a)= O(n_a)$, with q the number of grid cells within the radius of an atom, and n_a the number of atoms. Because every bucket of the structure with its contents requires copying to the GPU, there is an added cost factor of $O(n_b + n_a)$ for constructing the GPU-friendly data-structure before sending it to the GPU, where n_b is the number of hash buckets. A detailed description of the construction of this hash structure is given in section [3.6.3].

Constructing (and sending) the hash table with its contents is of complexity $O(n_b + n_a)$.

Theorem 2

For hash method A holds that the smaller the grid cell, the fewer atoms intersect or encapsulate it, making *k* smaller. However, a smaller grid cell size also means that more entries are added to the hash table. This holds because an atom should be mapped to every grid cell it occupies, a number that grows with decreasing cell size. So the construction time of the data structure on the CPU resulting from the grid cell size can be balanced against the data lookups of the particle query on the GPU resulting from the specific number of atoms in the grid cell.

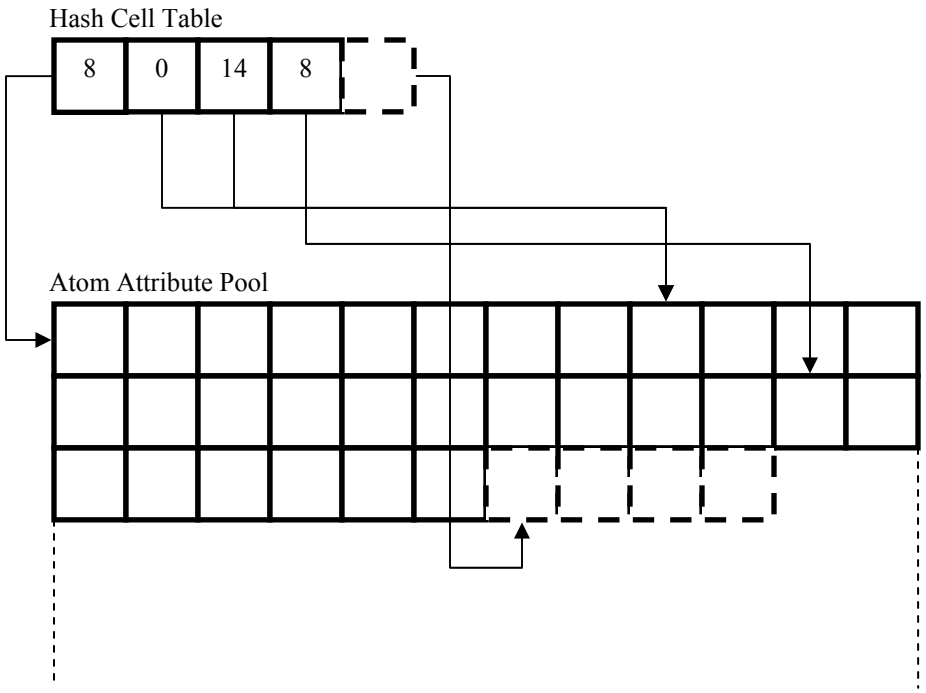
For hash method B a similar argument can be constructed, however in this case a decrease in grid cell size now implies an increase in the number of surface particle queries that have to be done. The construction of the hash data structure does not change. This holds because the influence sphere around a surface particle intersects more grid cells when the grid cell size is smaller. This implies in any case, the work is shifted around on the GPU, and there will be no CPU-GPU load balancing anymore.

The reason for choosing this data structure is the assumption that with a reasonable cell size the construction on the CPU does not take too much time, because in general it is linear in time complexity and can be scaled by choosing the right cell size and hash size. Furthermore, the access time of *O(k)* is very attractive, as long as the majority of atoms encapsulate the grid cell instead of intersecting it. An intersecting atom implies that a number of surface particles querying the grid cell have to evaluate an atom while actually residing outside the atom's radius.

3.6.3 Simulation Data Structure Implementation on the GPU

A data structure like the hash table above would be easy to implement in a classical CPU-memory model with libraries such as the STL. A fixed-size array of vectors would be an excellent choice for representing the fixed-size hash table with the corresponding buckets.

However, the GPU does not have any knowledge about these kinds of structures; the only available construct is a two-dimensional array of four-component floating point vectors accessed by texture coordinates. This requires the programmer to split the hash structure into two parts: a hash cell table, and a fluid atom attribute pool. The hash cell table replaces the fixed size array of buckets. The buckets are located in the attribute pool, forming a contiguous block of memory per bucket. The hash cell table is merely a reference to its corresponding bucket; a block of data in the attribute pool with a location and a size. For an example, see [Figure 18].



A Hash Cell Table indexing the Atom Attribute Pool, where the references are denoted by arrows, and the block sizes by the values in the Hash Cell Table

Figure 18

Querying the table involves a data lookup in the hash cell table, referred to as a "hash cell lookup" from now on. Only after this data lookup has been performed, the atoms can be accessed one-by-one by data lookups in the attribute pool. These data lookups will be referred to as "atom attribute lookups". Because the whole query still has *O(1)* complexity for the hash cell lookup and *O(k)* complexity for the atom attribute lookups, the overall query complexity does not change. As noted before, the visualisation method requires atom positions and velocities, so in this case two atom attribute pools indexed by the same hash cell table are desired.

Constructing the table can be done as follows: iterate over all cells in the hash to build the hash cell table of references into the data pool. At every cell, first append the corresponding bucket with atom attributes to the attribute pool. Now, the position and size of the bucket in the attribute pool is known and can be stored in the hash cell table. It is now clear why the complexity of [Theorem 2] involves an extra term of $O(n_b)$, because every cell needs to be visited in order to fill the hash cell table.

3.6.4 Choice of Nearest-Neighbour Algorithm for Visualisation

The second implementation problem is finding a method to establish densities and gradients thereof for surface particles, required for calculating the repulsion forces and density differences in [Equation 46] and [Equation 48] respectively. Both equations rely

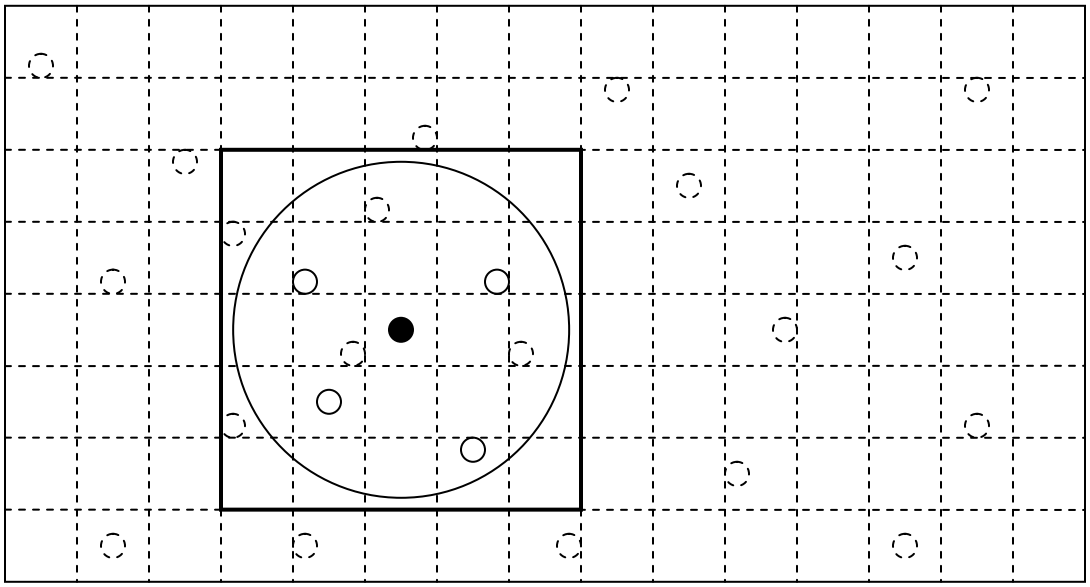
on SPH to interpolate field quantities in the neighbourhood of surface particles. Therefore, this section will construct a nearest-neighbour method for determining the surface particles that influence other particles nearby, in order to evaluate the value of field quantities at particle positions.

An obvious choice for such a method would be a spatial data structure, like the spatial hash in which the fluid atoms are distributed. However, building such a structure for the positions of the surface particles is very cumbersome on the GPU. The way a GPU works, is to treat pixels of a render target as independent units of calculation, where every calculation can write to only a very small (<10) fixed number of data locations. Because a spatial hash consists of buckets that grow dynamically while elements are added to it, a pixel cannot be equated with a bucket, since the pixel only has a finite amount of writeable memory attached to it. When an unknown amount of pixels are required for a single bucket, a dependency is created between the outcomes of mappings from different particle positions to the data locations they map to. This dependency implies that creating a hash structure on a GPU cannot be done in parallel (one pass), and that it probably has to utilise some kind of sorting mechanism in order to get the desired result. This has proven to be an expensive operation, which is outlined by [VIII]. The reliance on a sorting algorithm will also be an obstacle for the implementation of other spatial data structures.

Instead, the solution to this problem will be based on image-based rendering of the particle data. This imposes a constraint on the view frustum chosen for the nearest-neighbour algorithm: the participating elements – in this case the surface particles – should all be contained within the view frustum of the image-space projection. Otherwise their data will not be mapped onto the viewport, which means that no calculations can be applied to these elements.

The algorithm works in two steps. The first step makes sure the relevant information for every particle is present in the viewport. In the case of a nearest-neighbour search, this information is the position from every particle. To this end, the first step consists of rendering all surface particles to the pixel their position maps to in image-space. At this pixel location, the world-space position of the surface particle is stored. When two particles overlap, the front-most particle is stored and the other one discarded.

The second step performs the nearest-neighbour search on the data stored in image-space. Because each particle only influences other particles within a bounded sphere around its position, a sphere is used to identify the search area with. Envision that this sphere is rendered to image space. Every surface particle inside this sphere in world space would map to the same area as the sphere itself in image space. Simply put, all surface particles inside the sphere are also inside the disc formed by rendering the sphere into image space. The neighbours of a base particle can therefore be identified simply by rendering a disc around the particle, thereby evaluating at every pixel on the disc if there is a particle position stored at that location in image space. This can be done by reading from the image rendered in the first pass of the algorithm. If the same pixel in that image contains a particle position, a potential neighbour has been found. This is illustrated in [Figure 19].



An image-space projection of particle positions onto pixels in image-space, represented by the dots and dashed grid respectively. The black dot represents the base particle, around which a disc or square is drawn containing all pixels to which neighbouring particles might be mapped. All other dots represent the other particles, with white solid dots as neighbouring particles and dashed dots as particles outside the world-space sphere corresponding to the rendered disc.

Figure 19

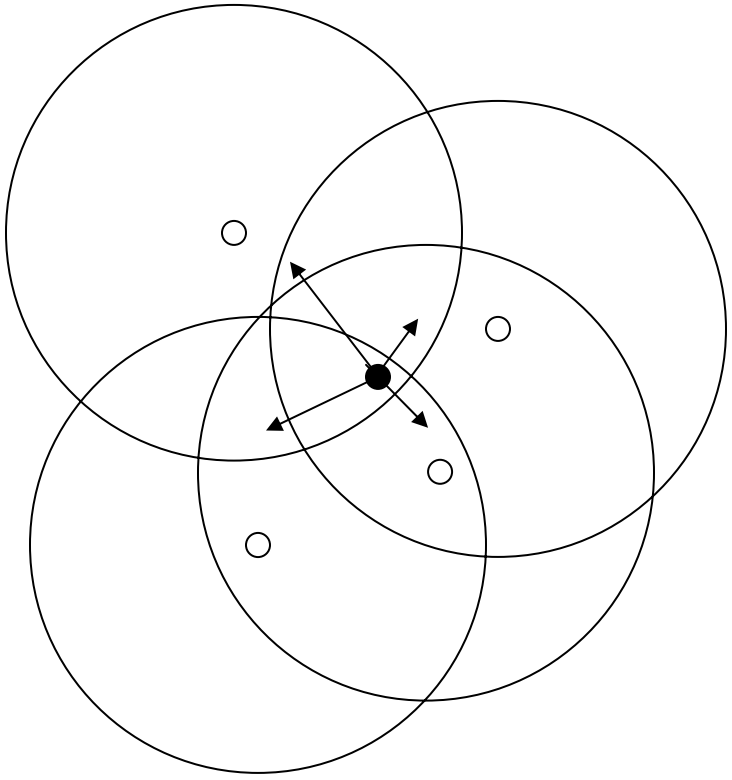
The image of the first pass stores world-space positions, so reading from it will yield the position of the particle that is a potential neighbour of the base particle around which the disc is being rendered. While the potential neighbour maps to the area of the disc around the base particle, it is still possible that its world-space position does not intersect the sphere around the base particle. It might be in front or behind the sphere in eye-space, occluding or being occluded by the sphere, without actually intersecting it. So before considering the particle a neighbour, the world space position of the particle read from the first-pass image has to be compared to the world-space position of the base particle, which can be provided as extra information while rendering the disc. When the two particles are close enough in world-space, they can be considered neighbours.

While the input and the method of the second pass have been explained, the output has not yet been discussed. The goal of the second pass is to calculate the repulsion forces and densities contributed by neighbouring particles. By rendering discs around surface particles, neighbouring particles can be identified in image-space and a repulsion force or density can be calculated, based on their distance in world-space and [Equation 46] or [Equation 48] respectively. So by rendering a disc around a base particle, the repulsion forces can be established that are contributed by the base particle to all its neighbouring particles, as in [Figure 20]. This repulsion force can again be stored at the location of the corresponding neighbour particle's location in image-space. Of course, one location in image space corresponding to a certain particle might be contained in many discs,

yielding many different values for repulsion forces and densities at a single position. These values can be accumulated by enabling additive blending while rendering the discs.

To recapitulate:

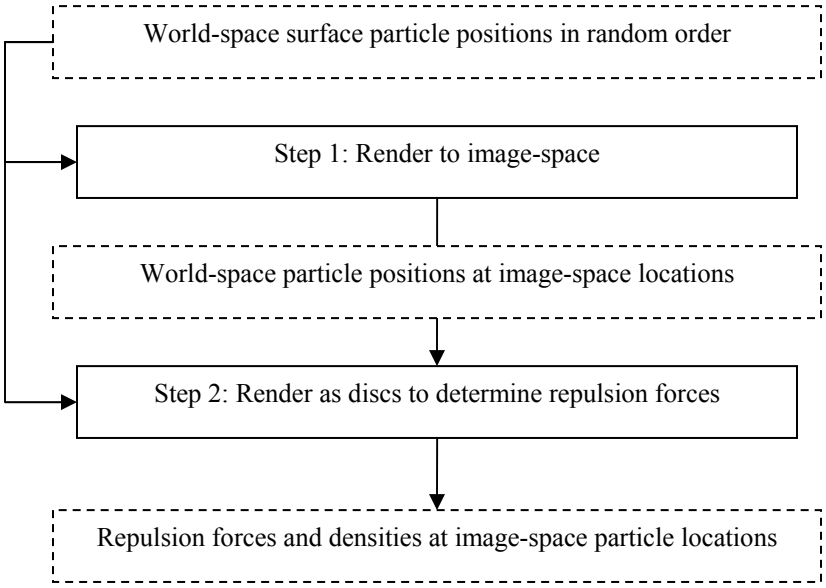
- Step 2 renders a disc around each particle's position in image space. For every particle position with disc, every pixel (fragment) of the image covered by the disc is considered separately and independently.
- At a single pixel *x* visited while rendering a disc, the image of the first pass is queried at *x* to find a neighbour particle's position.
- If a neighbour particle is found, a repulsion force or density is calculated based on the position of neighbour and base particle. The result is stored at pixel position *x* with additive blending enabled, to accumulate it with results from other particles acting on *x*.



The repulsion forces contributed to the base particle in black, by the neighbouring particles in white, through rendering a disc centered at every particle's position.

Figure 20

To summarise the above algorithm, an overview of the steps with the corresponding inputs and outputs is given in [Figure 21].



The two rendering steps are contained in solid boxes, while their input and output is specified in the dashed boxes.

Figure 21

In practice, the rendered discs of step 2 can be replaced by oriented quads or point sprites, as long as these primitives fully contain the disc that was originally intended to be rendered.

The repulsion forces and densities can be accessed again per particle, by storing the projection matrix at the time of rendering step 1 and 2 along with the image containing both quantities. Then, the repulsion force or density of a particle can be obtained by transforming the world-space position of the particle by using the stored projection matrix, yielding the corresponding image-based coordinate. The coordinate can then be used to extract the desired quantity from the image.

A final remark pertains to the choice of projection for rendering the information of step (1) and (2) in [Figure 21] to image-space. While both a perspective projection and an orthogonal projection will work, an orthogonal projection is preferred. A perspective projection maps a cone-shaped volume to image-space, while an orthogonal projection maps a cylinder-shaped volume. Both volumes have the same area of intersection at the near clip plane – namely of pixel size – which means that the cylinder-shaped volume is much smaller than the cone-shaped one. This implies much more culling for the cylinder-shaped volume, causing much less surface particle position overlap while mapping them to single pixels. Therefore, the orthogonal projection has an advantage over the perspective projection.

3.7 Test Results

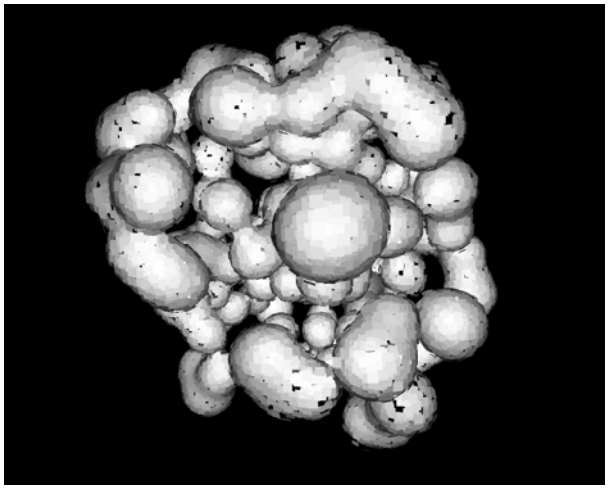
3.7.1 Test Program

For the following tests, a fluid simulation has been constructed to be used as test program. The application simulates a fluid according to the SPH principles and equations treated in the first part of the essay. On top of the simulation, the particle-based visualisation simulates is integrated into the application to simulate the movement of the surface particles. For different tests, different parameters are altered; these alterations will be indicated before all test results.

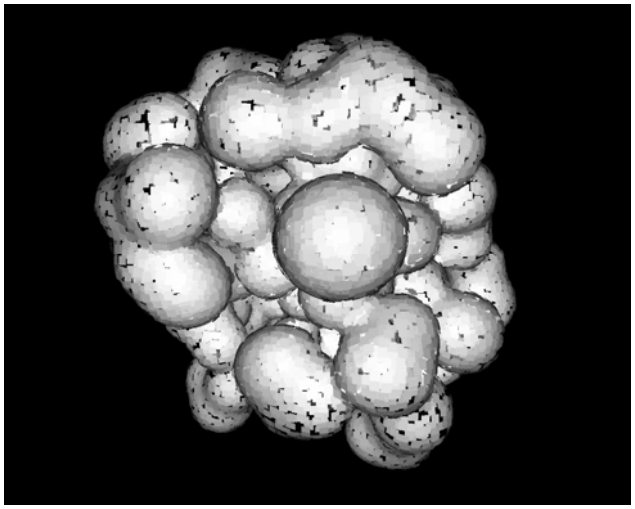
The tests themselves are divided into two parts, just like the visualisation method: the performance of the data structure of the fluid atoms for constraining the surface particles in section [3.7.2], and the visual effect of applying both local and global distribution to these particles in [3.7.3] and [3.7.4] respectively.

Because the tests will be based upon changing parameters of the visualisation, the fluid simulation is fixed, unless noted otherwise. Per default, the fluid simulation creates 100 fluid atoms and moves them around in space, without any external forces applied. The atoms have an influence (kernel) radius of 7 units, and they move towards a rest density of 0.05. Under these circumstances, the average distance between two fluid atoms has been measured to be 4.8 units.

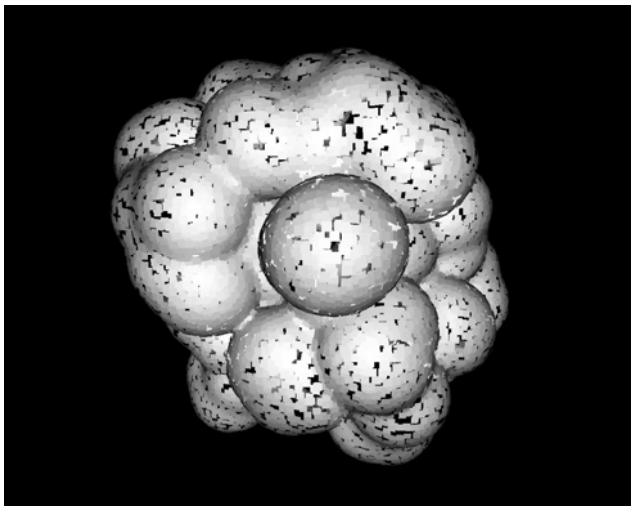
A fact of note is that for visualisation purposes the radius of the atom kernels is different from the radius used in the fluid simulation. This is done to be able to improve the performance of the algorithm; the smaller the atom radius used for visualisation, the fewer atoms influence the implicit function at a single location. This holds, because the atom radius for simulation remains unchanged, and therefore atoms will not move closer to each other. Of course, one should take care to make the radius big enough to still obtain a plausible looking fluid surface. In [Figure 22], the effect of three possible choices of atom radii for visualisation is visible, without changing the atom radius used during simulation. On the basis of these images, the default radius for visualisation is chosen to be 3 units wide.



The first out of three choices for the radius used at visualisation of the atoms, with a radius of 0.7 units.



The second out of three choices for the radius used at visualisation of the atoms, with a radius of 1.5 units.



The third out of three choices for the radius used at visualisation of the atoms, with a radius of 3.0 units.

Figure 22

3.7.2 Hash Parameters

The following test evaluates the choice of different parameters for the spatial hash structure. First, the performance of the hash itself will be investigated by changing its size. Then, different grid cell sizes are evaluated to see what the impact on the performance is dividing space into smaller or larger cells.

Both surface distribution mechanisms for particles are turned off, to prevent them algorithm from having an influence on the performance measured in the upcoming tests. Furthermore, the surface is approximated by 65536 particles. This number has been chosen to be as high as possible while allowing the application to run at a decent framerate, to be able to observe the impact on the performance of the particle simulation that uses the structure as well.

The hardware used for the following tests is a GeForce 6800GT with 256MB of video memory and driver version 81.98, an Intel Pentium 4 3.2 Ghz with HyperThreading, and 2GB RAM.

Hash method A is the first method to be tested for performance. To recapitulate, this method adds a fluid atom to the hash multiple times at construction by scanning the entire volume, and then queries the table just once for every particle position in the surface visualisation. See [Figure 17].

The problem statement is as follows: find the best choice for both the hash size and the size of the grid cells, such that the number of elements per bucket is minimal. Without further constraints the answer would be trivial: take an as big as possible hash with as small as possible cells, as long as the amount of memory allows it. Therefore another constraint has to be posed on the work the CPU has to do; it should not become the bottleneck of the simulation. In fact, a configuration is desired where the performance gain on the GPU by a further increase in hash size/decrease in grid cell size is offset by the decrease in performance on the CPU and memory caused by constructing and transporting the extra hash data associated with the increased size.

First off, the goal is to find a range of suitable hash sizes that do not impose a bottleneck at construction of the data structure on the CPU. So the hash size is altered, while keeping the cell size fixed. In terms of [Theorem 2], this implies that n_b is altered, while n_a is kept constant. The cell size is chosen to be 1.5 units wide, so that for every atom the following number of grid cells need to be filled:

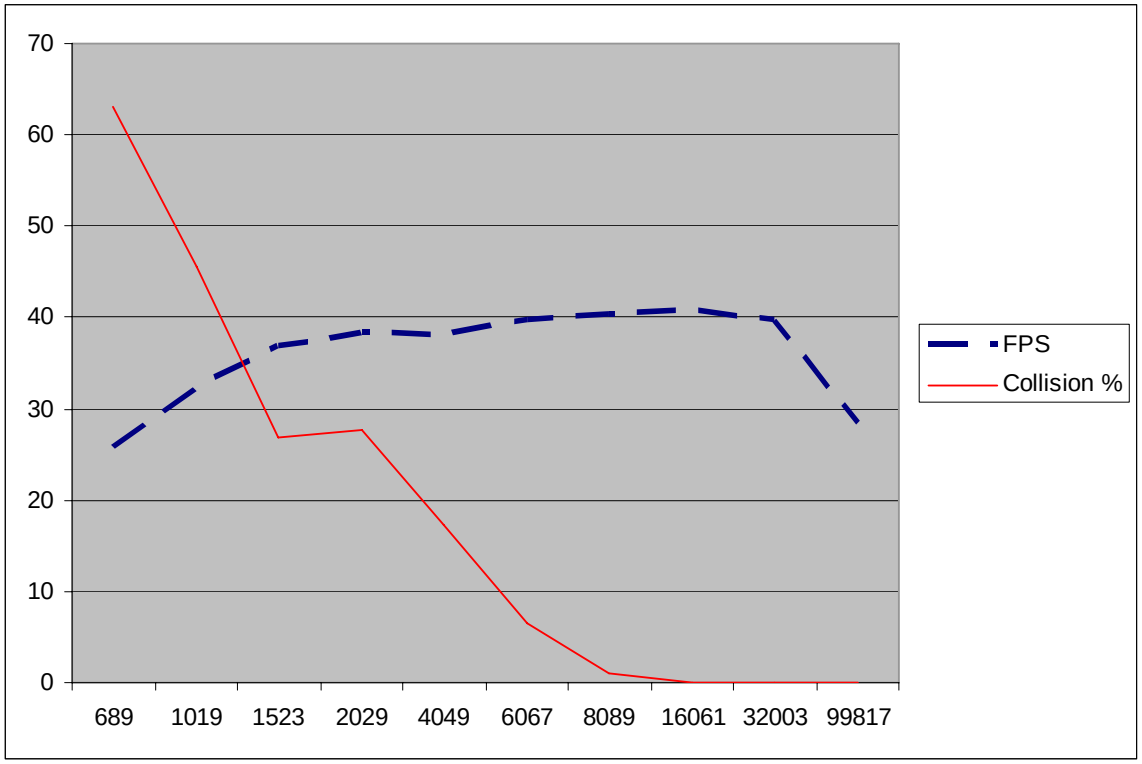
> *Grid cells encapsulated or intersected by the bounding box of an atom's influence volume* $= (radius * 2 / cell\ size + 1)^3 = (3.0 * 2 / 1.5 + 1)^3 = 5^3 = 125$

For 100 atoms, this implies the insertion of 12500 elements in the hash. In our simulation, all these elements covered a total amount of around 1800 grid cells when in a rest state. Since these unique grid cells are mapped to hash buckets, a good starting point for the hash size of this simulation would be somewhere close to 2000 buckets.

The trade-off in the following test is the extra construction time induced by an increase in hash size, against the reduced cost of the surface simulation because of the reduced number of hash collisions. As can be derived from the above example, the number of filled buckets in the hash is much smaller than the total number of data elements inserted into the hash. Expressed in terms of our hash construction complexity [Theorem 2], the above implies that $n_b < q * n_a$, with q the number of grid cells in a single atom's bounding box. Therefore it is not expected that increasing the hash size will have a negative influence on the performance of the algorithm. Also, it is expected that an increase in hash size will not decrease the number of collisions anymore at a certain number of buckets when the distribution of the added atoms is ideal. To see if these expectations are correct, both the number of hash collisions and the time of constructing and transferring the data will be measured in the following tests.

Hash Size	689	1019	1523	2029	4049	6067	8089	16061	32003	99817
Avg. FPS	25,89	32,38	36,78	38,32	37,98	39,76	40,24	40,65	39,71	28,59
# of Unique Grid Cells	1770	1770	1757	1741	1787	1704	1715	1766	1741	1807
# of Nonempty Buckets	654	964	1286	1259	1479	1593	1698	1766	1741	1807
Collision %	63,05	45,54	26,81	27,69	17,24	6,51	0,99	0,00	0,00	0,00

Table 1



The frames per second (dashed blue line) and collision percentage (thin red line) plotted against the hash size.

Figure 23

As expected, [Table 1] shows that increasing the hash size does not limit the performance of the simulation, as long as the size is not chosen to be much larger than n_a. All hash sizes above a bucket number of 2029 perform more or less the same in the case of 1700 unique grid cells, since the distribution causes few collisions anymore. The hash size can be increased to 32003 elements before the construction of the hash table becomes a bottleneck of the total simulation performance, which is far above $q*n_a$ in this case. However, hash sizes which are too small are much more of a threat to the overall performance. The collision percentage increases for hash sizes below 2029 buckets, meaning a higher workload for the surface visualisation on the GPU. This is reflected in the framerate at the left end of [Figure 23], which compares the framerate to collision percentages.

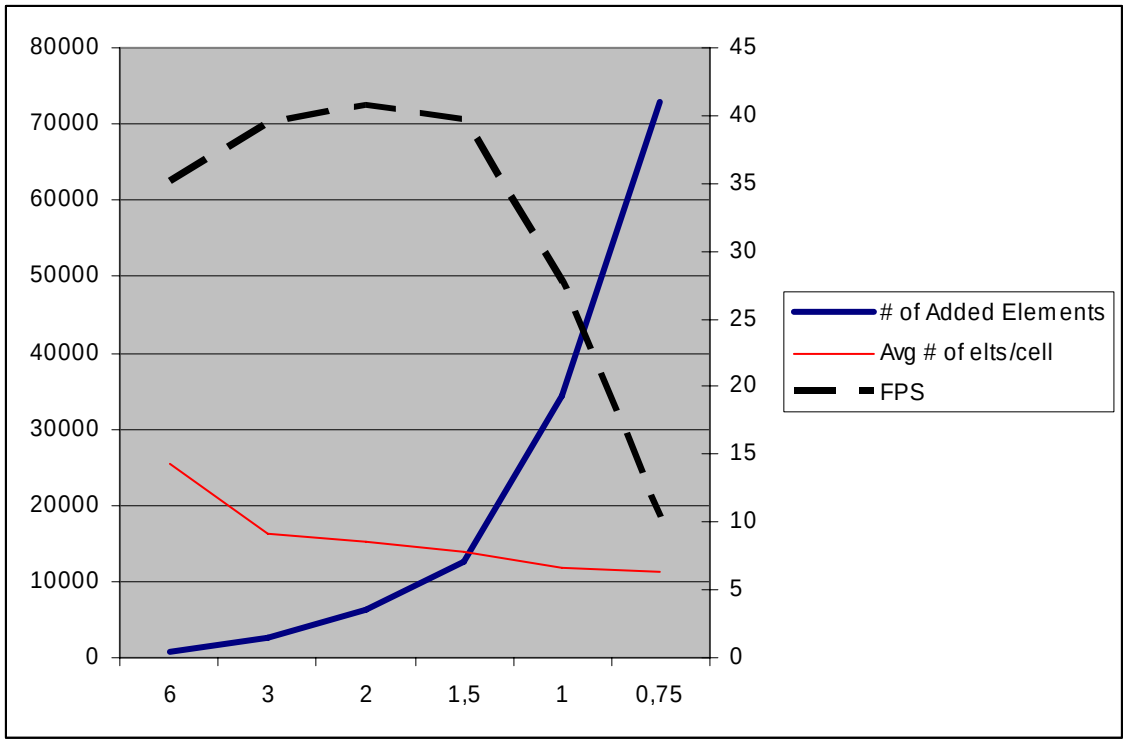
The next step in the performance evaluation is to change the grid cell size. Changing the grid cell size alters the number of elements added to the hash per atom, and inserting atoms is part of the hash construction. Furthermore, changing the grid cell size also changes the number of atoms that intersect one cell. The smaller the cell size, the better it approximates the positions inside it, and the fewer atoms will be added to the cell. So the visualisation will have to perform less atom attribute lookups and will be quicker as a result. Of course, counteracting this performance gain will be the increased number of added elements increasing the insertion time. The goal of the next test is to see how such a trade-off turns out for different grid cell sizes.

Because an increased number of element insertions per atom also implies that an increased number of unique grid cells will be added to the hash table, the hash table size is increased to roughly correspond with the number of unique grid cells, and keep the number of collisions low. As long as $n_b < q* n_a$, [Theorem 2] dictates that increasing the hash size will not impose a bottleneck during construction of the hash table, while [Table 1] shows that the extra construction time is offset by the performance gain caused by reducing collisions.

Apart from the changes in hash size and grid cell size, the conditions of the next test are the same as for the previous one.

Cell Size	6	3	2	1,5	1	0,75
# of Added Elements	800	2700	6400	12500	34300	72900
# of Unique Grid Cells	56	299	815	1704	5269	11626
Hash Size	631	1523	4049	6067	16061	39019
# of Nonempty Buckets	56	297	745	1593	5169	11533
Collision %	0,00	0,67	8,59	6,51	1,90	0,80
Avg # of Elts over Nonempty Grid Cells	14,29	9,03	7,85	7,34	6,51	6,27
Avg # of Elts over Nonempty Hash Cells	14,29	9,09	8,59	7,85	6,64	6,32
Avg FPS	35,09	39,38	40,73	39,76	27,71	10,37

Table 2



The number of added elements (thin red line), average number of elements per cell (thick blue line) and frames per second (dashed black line) plotted against the cell size. The left vertical axis shows the scale of the first attribute, while the latter two are scaled by the right vertical axis.

Figure 24

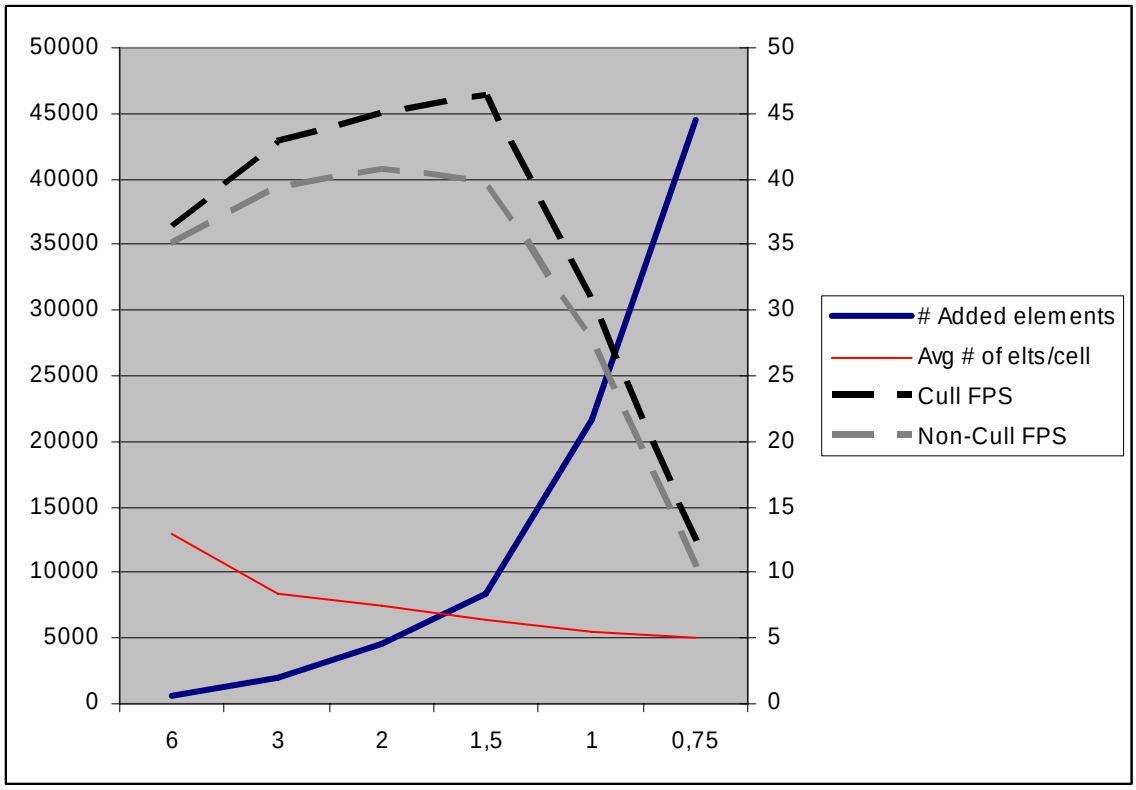
[Table 2] shows situations in between two extremes. At the left end of the spectrum, the bottleneck is purely the surface particle simulation on the GPU, dependent on the average number of elements over nonempty hash cells. This can be illustrated by turning off the surface visualisation itself or eliminating the atom position lookups in the simulation. Doing the latter improves the framerate towards 93.55 fps. At the right end of the spectrum, the framerate is limited by the insertion of elements into the hash, performed by the CPU. Eliminating atom position lookups yields a framerate of 11.13 fps, a negligible difference.

The results show that for big cell sizes, the average number of elements per grid cell increases. Because initially the surface simulation is the bottleneck of performance, a speedup is gained by decreasing the size of the grid cells, offloading the work the surface simulation has to do. However, at the same time the number of added elements increases drastically with $O(n^3)$. This is reflected in a bottleneck shift: instead of the simulation of surface particles, the insertion of the elements for every atom at the construction of the hash becomes a bottleneck. So the bottleneck shifts from the GPU to the CPU. This shift can be detected by looking at the point where the framerate starts to decrease; at this point the performance gained by offloading the surface visualisation does not offset the performance penalty by the extra work of the hash construction anymore. [Table 2] shows that this bottleneck shift is located around a grid cell size of 1.5.

While [Table 2] shows an average number of atoms per grid and hash cell, this does not have to be equal to the average or median number of atoms accessed by a surface particle on the GPU. This is caused by the uneven distribution of particles in space; indeed, they tend to move to the fluid surface. Further investigation showed that in the case of grid cell size 1.5, the median number of elements accessed per surface particle was 12 atoms, while the median number of atoms actually influencing the particle was 5. The cause of this difference is the bounding volume of the atom influence sphere; the sphere is approximated by an axis aligned bounding box, which means that some grid cells to which the atom is added are not intersected by the influence sphere at all. If one wants to minimize the amount of work done by the surface visualisation, it is possible to perform a sphere-box intersection test every time a fluid atom is added to the hash at construction of the hash table, to be certain the fluid atom has influence inside the grid cell. The construction of the hash table is performed on the CPU, so this change would shift some work from the GPU to the CPU. This implies the bottleneck shift from the results of [Table 2] will change as well. So it is interesting to know if this change has any influence on the bottleneck, and the overall framerate.

Cell Size	6	3	2	1,5	1	0,75
Added elements reduction	12%	22%	29%	33%	37%	39%
# of Added Elements	664	2052	4544	8375	21609	44469
# of Unique Grid Cells	51	246	649	1361	3976	8754
Hash Size	631	1523	4049	6067	16061	39019
# of Nonempty Buckets	51	246	614	1322	3956	8742
Collision %	0,00	0,00	0,05	0,03	0,01	0,00
Avg # of Elts over Nonempty Grid Cells	13,02	8,34	7,00	6,15	5,43	5,08
Avg # of Elts over Nonempty Hash Cells	13,02	8,34	7,40	6,34	5,46	5,09
Avg FPS	36,37	42,76	44,93	46,38	30,62	12,27

Table 3



The chart is showing the same parameters as [Figure 24], but now with culling enabled. The number of added elements (thin red line), average number of elements per cell (thick blue line) and new frames per second (dashed black line) are plotted against the cell size. The old number of frames per second (dashed grey line) is added as reference. The left vertical axis shows the scale of the first attribute, while the latter three are scaled by the right vertical axis.

Figure 25

As visible in [Table 3], the framerate has improved across the whole table, even in CPU-limited situations. Apparently, the performance gain by building a hash structure with around 30% less elements has more effect on CPU performance than the extra work done by the sphere-box intersection test. The bottleneck shift has moved as well, to a cell size of 1; a cell size of 1.5 is not CPU-limited anymore.

Next, hash method B is tested for performance. Combining the results of the previous test – namely that the simulation is GPU-limited because of atom position lookups – with the nature of the hash method compared to method A – offsetting the tasks the processor has to perform at construction of the hash table by demanding more work of the GPU in the form of texture lookups – suggests that the performance will not improve. To test this hypothesis, the same approach can be taken towards method B as towards method A. The grid cell size could be decreased or increased, resulting in a larger respectively a smaller number of grid cells to be queried by the surface visualisation.

However, in this case the test can only be performed with a single grid cell size, namely 2*radius. This means the influencing atoms can be found by visiting 8 grid cells, namely all cells containing one of the corners of the bounding box with size 2*radius around a surface particle.

To understand why the grid cell size cannot vary, it is necessary to explain how the volume around a surface particle is scanned and queried. Consider the axis aligned bounding box around the particle volume with a size of 2*radius. It is impossible to query every point within the volume to see what grid cells are intersected, for that would mean an infinite number of queries. Therefore the query starts at a corner of the bounding box. In every direction steps are taken equal to the grid cell size. This ensures that at every step in a certain direction, the starting position relative to the current grid cell is exactly the same as the new position relative to the neighbouring grid cell in the direction of the position change. Therefore, no grid cells encapsulated or intersected by the bounding box are skipped over by the collection of queries, and none are visited twice. Of course, when the size of the bounding box in any direction is not divisible by the grid cell size (i.e. it does not yield an integer), the number of queries need to be rounded off to the next integer. For reference, the atoms in hash method A are added to the hash table by scanning their volume in the same way.

Say, the grid cell size fits more than one time into the size of the surface particle bounding box. We start in the corner of the bounding box and perform a minimum of two extra queries in every direction. This means that at least 3*3*3=27 positions need to be queried, yielding a minimum of 27 hash cell lookups. This implies a minimum of 27 data lookups before the atom attribute lookups have even started. The tests of hash method A already taught us that such a large number of lookups per particle is too much. As a result the grid cell size has to be at least the size of the bounding box, namely 2*radius. Only the corners of the bounding box are used for hash table queries now, yielding only 8 hash cell lookups.

Cell Size	6
# of Added Elements	100
# of Unique Grid Cells	18
Hash Size	2029
# of Nonempty Buckets	16
Collision %	0,11
Avg # of Elts over Nonempty Grid Cells	5,56
Avg # of Elts over Nonempty Hash Cells	6,25
Avg FPS	6,22

Table 4

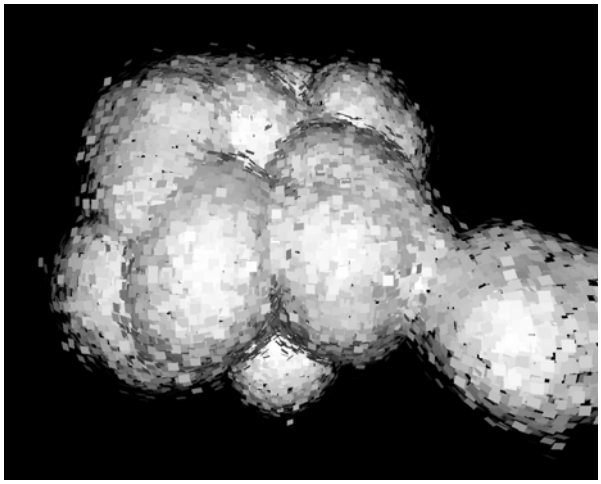
As visible in [Table 4], the average number of elements for a single query of a particle bounding box's corner yields around 6 data lookups. Therefore, the average number of data lookups for all queries for a single surface particle is maximally 6*8 = 48. This is still larger than hash method A, and the suspicion that the larger number of data lookups would have an adverse effect on the framerate is correct; an average framerate of 6.22 is merely a sixth of what has been observed for the same cell size in hash method A.

3.7.3 Particle Repulsion

In this section, the local particle distribution algorithm is studied in isolation. It will concentrate on the repulsion of particles on the fluid surface with global distribution turned off. In section [3.7.4] the global particle distribution algorithm is added.

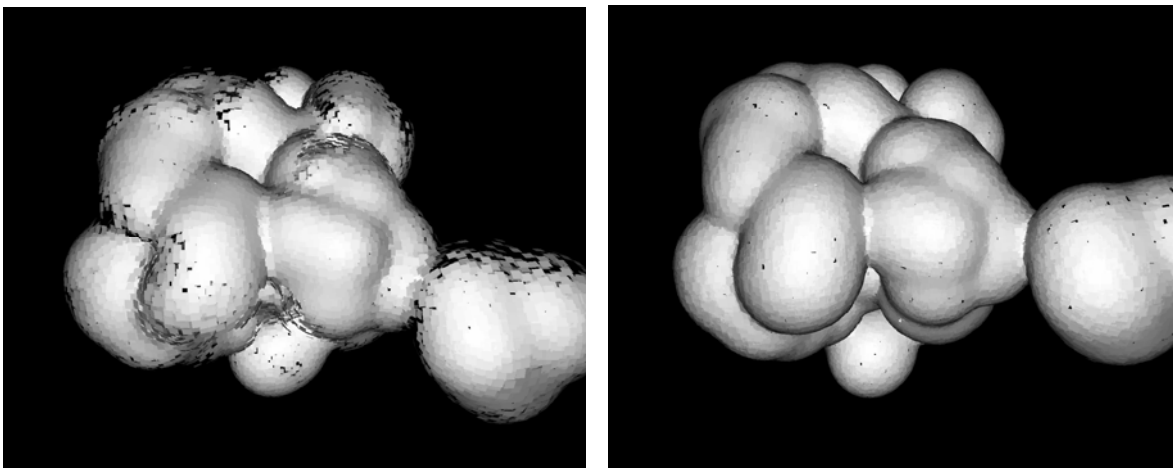
According to [Equation 46], the parameters of repulsion are the scaling factor of the repulsion force and the radius of the Smoothing Kernel. Increasing the radius of the Smoothing Kernel directly implies an increase in rendering time, so it is advisable not to make it much larger than the actual surface particle drawn. The reasoning behind this is that two surface particles do not have to repel each other as long as they do not overlap when they are actually rendered. Likewise, the Smoothing Kernel should not be smaller than the drawn surface particle either. So the size of the Smoothing Kernel is fixed, as long as the size of the particle is fixed. An evaluation of particle sizes is given at the end of this section.

Unlike the Smoothing Kernel size, the repulsion scaling factor can be chosen independently. The higher the scaling factor, the more two particles repel each other. However, this repulsion is accompanied by an increase in particle velocity. Therefore, a high repulsion factor makes particles restless and the surface of the fluid instable. A low repulsion factor slows down the distribution of particles over the surface. The effect of this coupled with dampening of the surface particle velocity, is that surface particles may never reach an even distribution and gaps may remain. These cases are illustrated by [Figure 26], [Figure 27] and [Figure 28]. The key is to find the ‘correct’ repulsion force: not too strong and not too weak. Unfortunately, finding this correct force remains an unknown so far; human interference is necessary to tweak the force to get the effect that behaves correctly.



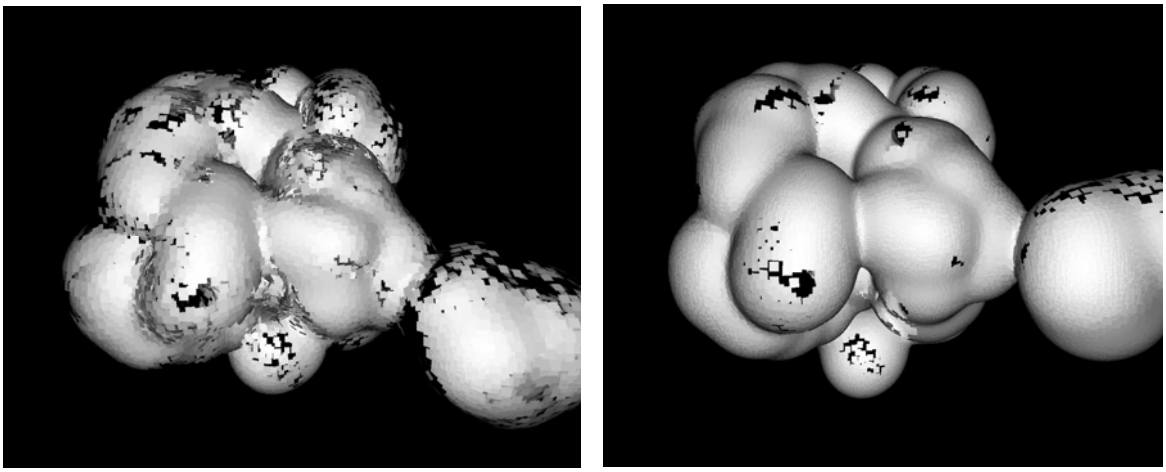
One out of three different situations with different repulsion forces for non-moving fluid atoms. The first situation shows a high repulsion force causing a restless fluid surface, with particles moving quickly.

Figure 26



One out of three different situations with different repulsion forces for non-moving fluid atoms. This is the second situation, showing a lower repulsion force than the first. The first image is taken after five seconds, while the second one is taken after fifteen seconds.

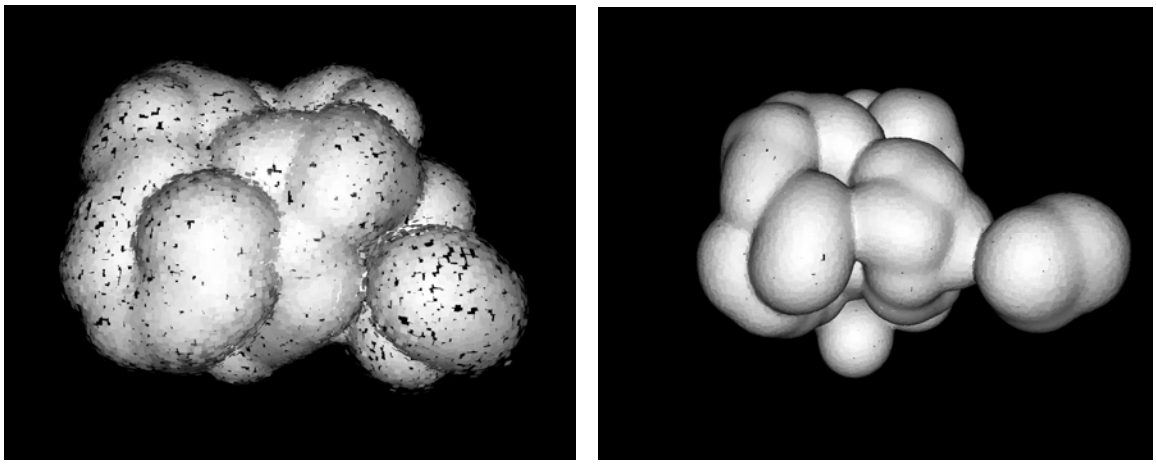
Figure 27



One out of three different situations with different repulsion forces for non-moving fluid atoms. This is the third situation, showing the lowest repulsion force from all three. The first image is taken after five seconds, while the second one is taken after fifteen seconds.

Figure 28

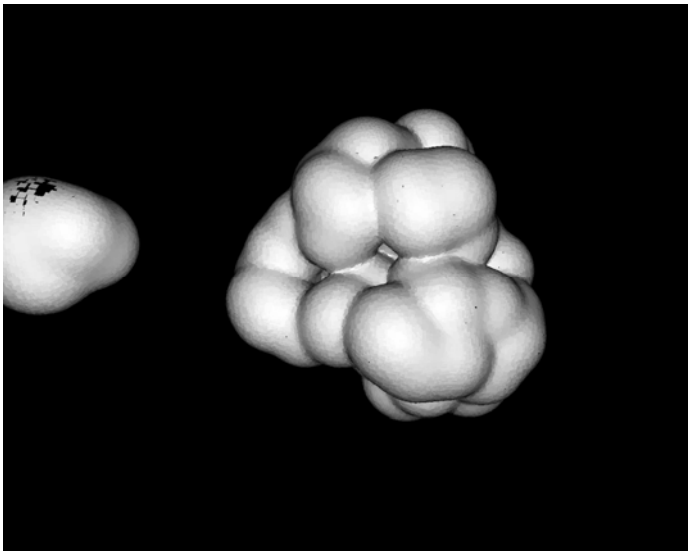
So far, only the parameters of the particle repulsion method themselves have been treated. In [Figure 29], the effect as a whole is demonstrated and compared to moving the particles at a fixed speed in random directions across the fluid surface. The images show a situation that has stabilised after some time, where the fluid atoms have come to rest and the particles had some time to distribute themselves across the surface.



The left image shows the fluid simulation in which the particles move with random speed across the surface. The right image shows the fluid simulation in which the surface particles have had time to distribute themselves evenly across the surface. The fluid atoms do not move in these cases.

Figure 29

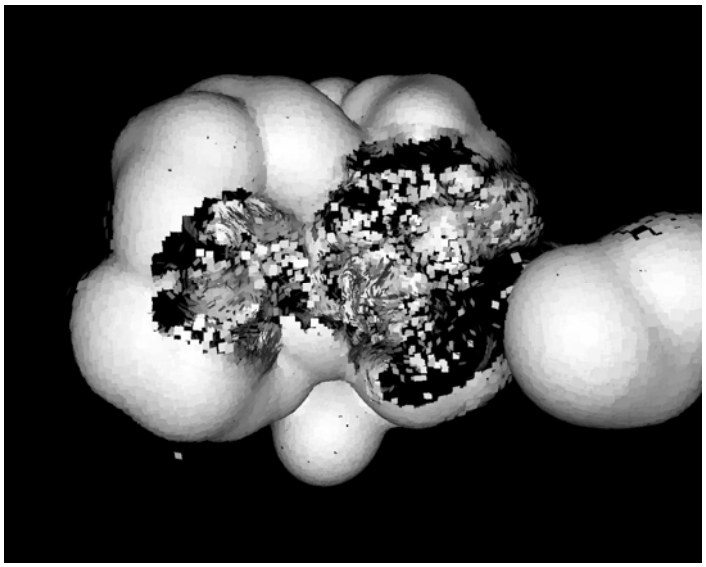
It is visible that the distribution works; the particles distribute themselves evenly across the surface. This does not change when moving the camera, as can be seen in [Figure 30], where the camera has immediately moved to the back without allowing the particles to distribute themselves even further:



The same fluid as in [Figure 29], but now the camera is switched to the back shortly before making the snapshot.

Figure 30

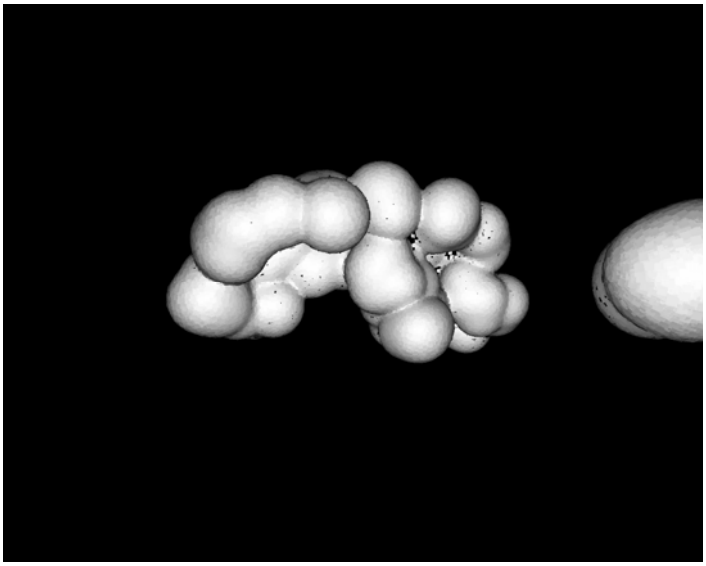
However, the above describes a situation in which the fluid atoms do not move. Should one choose to apply an external force to the fluid – like a bullet hitting the surface – the fluid atoms will move apart, requiring the surface particles to keep the surface covered as much as possible. This is illustrated by [Figure 31], which shows that gaps will form in the fluid surface immediately after the collision.



The fluid surface after a bullet has hit the fluid

Figure 31

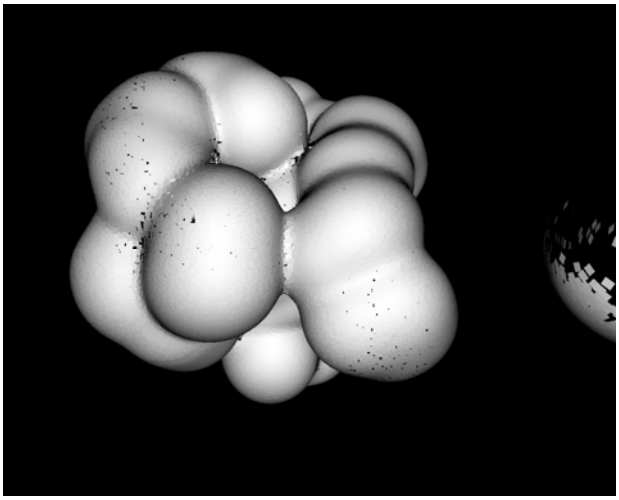
These gaps are the result of the surface particles not having converged to an even distribution yet, which they will have to do as quickly as possible. In this case the whole fluid blob starts to move, which introduces another problem: the surface particles have to move along with the fluid atoms, while distributing themselves evenly across the surface at the same time. This is where [Equation 37] does its job, establishing the average velocity of the environment of a fluid particle and moving this particle accordingly. [Figure 32] shows that the method works out; after a while, the surface particles are evenly distributed again.



The fluid surface moving away from the viewer

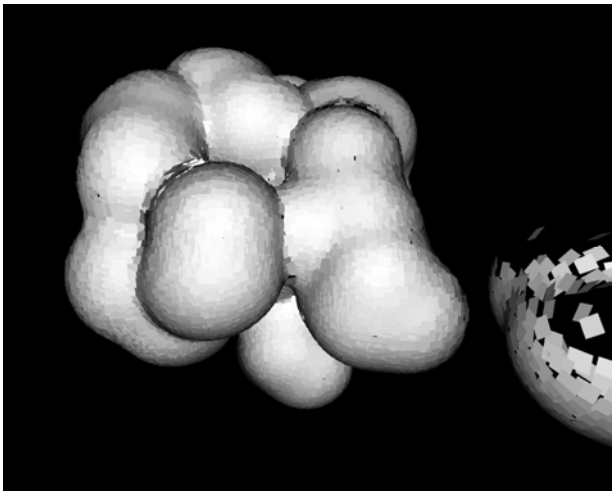
Figure 32

Surface particle sizes can be varied as well to improve coverage of the fluid surface, but this comes at the cost of a loss of approximation precision. However, the improved coverage implies one could do with simulating fewer particles as well, so the performance might improve. To test this, three situations are chosen with varying amounts of particles and particle sizes, where choosing more particles implies a smaller particle size. The sizes are chosen such that a good coverage of the fluid surface is achieved when the particles are distributed evenly across the surface. The results are shown in [Figure 35].



One out of three images showing the fluid surface rendered at various levels of detail. This first picture has the highest resolution, with low performance due to the high number of simulated particles. The framerate is 12 fps in this case.

Figure 33



One out of three images showing the fluid surface rendered at various levels of detail. This second picture has a resolution four times as low as the first, with an equivalent increase in particle size. This increases the performance of the visualisation compared to [Figure 33]. The framerate is 22 fps.

Figure 34

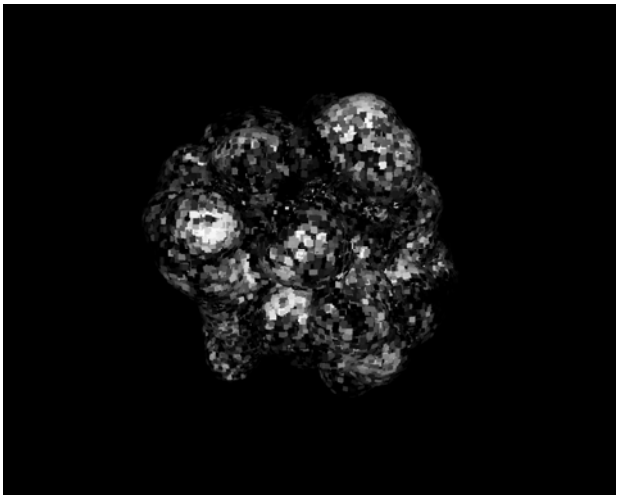
One out of three images showing the fluid surface rendered at various levels of detail. This third picture has a resolution four times as low as the second, with an equivalent increase in particle size. This increases the performance of the visualisation compared to [Figure 34]. The framerate is 30 fps.

Figure 35

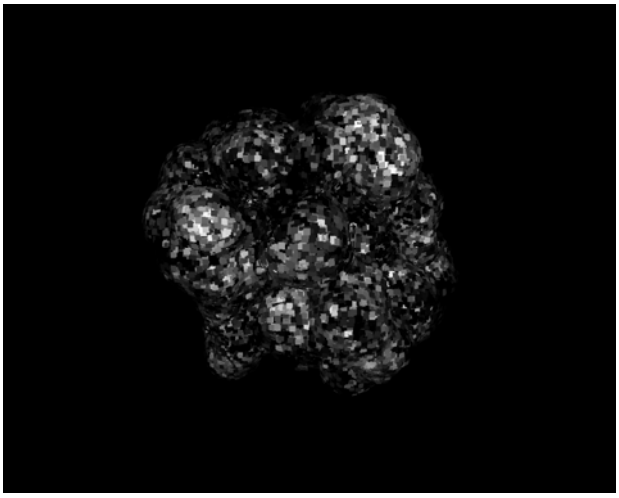
3.7.4 Particle Dispersion

Section [3.7.3] treated the particle repulsion, and it was clear that the surface particles still required a considerable amount of time – about ten seconds – to distribute evenly across the surface of the fluid. The local particle repulsion technique which has been used up until now does not seem to be sufficient for quick particle distribution. Therefore, this section investigates the effect of global particle distribution used on top of the existing local distribution technique, for quicker distribution of particles across the fluid surface.

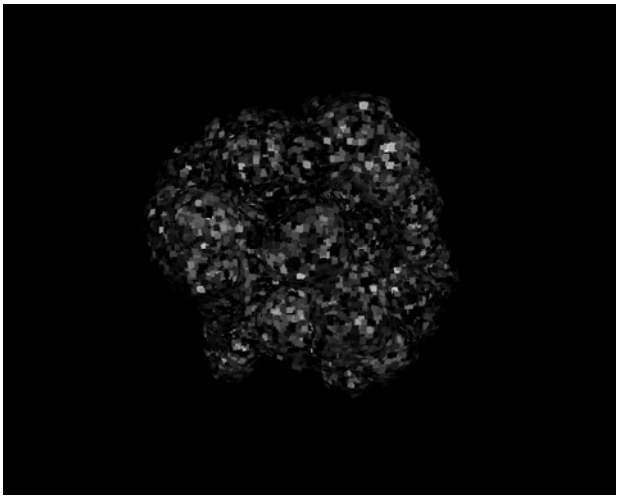
The effect of the global distribution algorithm can be depicted by images showing the density distribution of the particles after a few seconds. The density distribution is observed on the basis of [Equation 48]. This equation provides a threshold for changing particle positions by comparing their densities. While it appears that this threshold is not to be changed, the images of [Figure 36] show that it does have a large influence on global particle distribution. The global distribution algorithm should eliminate large clusters of particles, and [Figure 36] shows how well the algorithm performs with varying choices for the threshold.



Density of the fluid with the global distribution algorithm turned off, after two seconds of distribution.



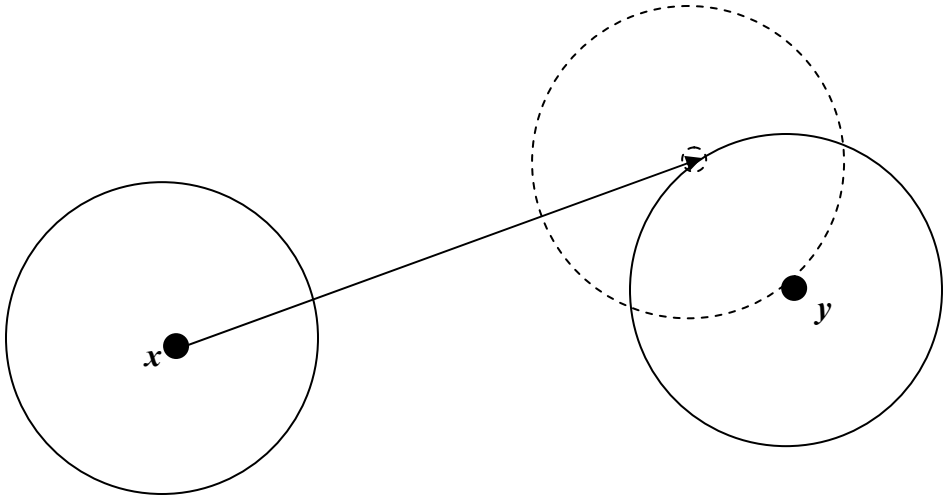
The global distribution algorithm on with a threshold of *cW(**0**,h')*, after two seconds of distribution.



The global distribution algorithm on with a threshold of *0.5*cW(**0**,h')*. The image was taken after two seconds of distribution.

Figure 36

Remarkably, [Figure 36] to [Figure 36] shows that choosing $cW(\mathbf{0},h')$ for the original threshold of repositioning the particle is too high; the density is not distributed much better than without the global distribution algorithm. Choosing half the threshold eliminates that problem, the particles distribute much better in this case. However, the surface is more restless now, and the density difference $\Delta\sigma(\mathbf{x},\mathbf{y})$ between position $\mathbf{x}$ and $\mathbf{y}$ before a particle position change can be smaller than afterwards. This can be overcome though, by placing the particle not exactly at the position of its comparison particle, but somewhere on the influence boundary. This is illustrated in [Figure 37].

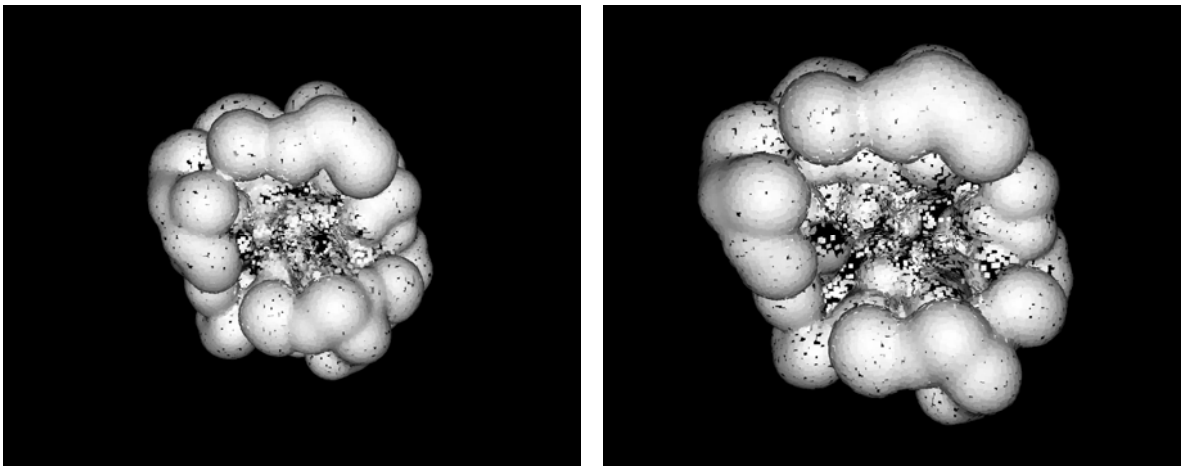


The same situation as in [Figure 13], with the neighbour particles' influence spheres omitted. This time, the particle is moved to a position on the boundary of the comparison particle. It causes less of a change in density difference between position $\mathbf{x}$ and $\mathbf{y}$ than the original method of moving the particle at $\mathbf{x}$ to position $\mathbf{y}$.

Figure 37

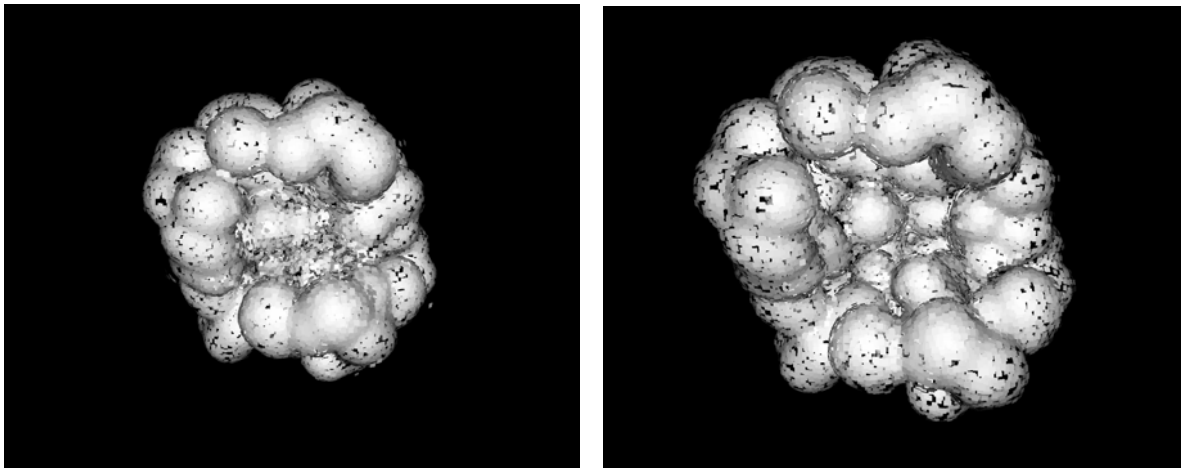
Now, the difference in $\Delta\sigma(\mathbf{x},\mathbf{y})$ before and after the position change is reduced, compared to positioning the particle exactly at $\mathbf{y}$.

Finally, the effect of global distribution is ready to be visualised for a restless surface. To this end, the situation of a bullet hitting the surface is revisited, with [Figure 39] and [Figure 39] showing the difference with global distribution on and of at different moments in time.



The situation of a bullet hitting the surface, depicted one second after the moment of impact (left images) and 2.5 seconds after the moment of impact (right images). These images show what happens when global distribution is turned off.

Figure 38



The situation of a bullet hitting the surface, depicted one second after the moment of impact (left images) and 2.5 seconds after the moment of impact (right images). These images show what happens when global distribution is turned on.

Figure 39

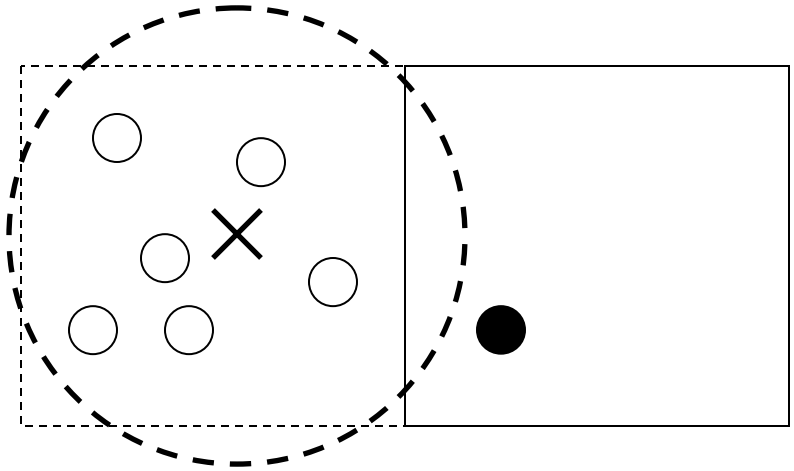
The images of [Figure 39] and [Figure 39] show that while the global distribution still creates a more restless surface, the distribution of particles is far better than in a situation without. The advantage of the global distribution is that the restlessness can be tuned to the desired level, or hidden by choosing a more sophisticated rendering method.

3.8 Additions to the Surface Visualisation

3.8.1 Approximating the Surface's Implicit Function

A possible solution towards reducing the atom data required for calculating the terms of [Equation 36] is to employ approximation for calculating the contribution of multiple fluid atoms to the implicit function *F*. This can be illustrated by describing a variation on hash method B. Recall that in hash method B, a fluid atom is added to the hash table just once by its position, requiring the surface simulation to query a volume around each surface particle to find all atoms influencing that particle. When choosing a hash cell size of 2*radius, sampling the corners of the bounding box around the particle with size equal to the hash cell size would be sufficient; it ensures that the hash cell containing the surface particle is queried, along with all neighbouring hash cells (partially) within the bounding sphere of the surface particle.

The most important factor limiting performance in the previous test was the sheer amount of fluid atoms such a query could yield. Therefore, the goal of the following section is to reduce the data lookups induced by the query. To accomplish this, a method is developed to approximate the contribution of a certain subset of all fluid atoms to the implicit function. The approximation is based on a certain property of fluid atoms occupying the same hash cell, namely that these atoms are always positioned close to each other. Now take the grid cell c_1 of the evaluation position belonging to a surface particle, and an arbitrary neighbour cell c_2. One can always choose a point p in space such that all the fluid atom positions in c_2 are closer to p than any surface particle in c_1. This situation is highlighted in [Figure 40], and it is exactly the property required for the applicability of the approximation technique described in section [3.8.2]: Multipole Expansion.



The surface particle (black) with fluid atoms (white) in the main grid cell and the neighbour grid cell respectively. The arbitrarily chosen position is denoted by a cross, and is not necessarily located in the neighbour grid cell. All fluid atoms are closer to the cross than the surface particle. If required, the arbitrary position can be placed infinitely far away from the division between the grid cells, so that the circle around it approximates the plane dividing the grid cells.

Figure 40

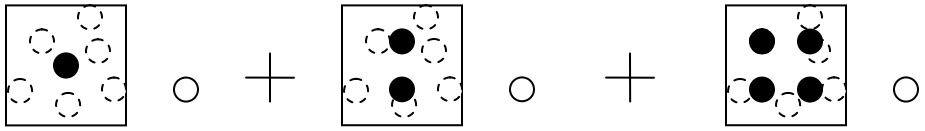
The property described above applies to the collection of atoms in neighbour cells of the grid cell encapsulating a surface particle, but not for atoms within the surface particle's grid cell itself. Therefore, Multipole Expansion can be employed for the atoms in every separate neighbour cell, while the fluid atom positions in the surface particle's own cell still have to be evaluated one at a time. The consequence is that not every fluid atom from every neighbour cell needs to be evaluated anymore. This will hopefully reduce the number of data lookups performed.

3.8.2 The Fast Multipole Method

As explained in section [3.8.1], an approximation method is required for evaluating the implicit function formed by atoms in a grid cell from a position in one of the neighbour cells. Such a method is already used in atom simulations to reduce the quadratic complexity of the calculation of forces between atoms. The technique is based on the Fast Multipole Method, of which an overview is given in [V]. A similar kind of overview is given in this section and [Appendix A], to provide a mathematical insight into the applicability of the method towards the problems faced by our current surface visualisation.

A simple summary of the FMM technique boils down to the approximation of the implicit function defined by all atoms of a single grid cell by calculating an "atom expansion". This expansion technique will try to approximate the implicit function with

increasing accuracy according to the number of terms (orders of approximation) one chooses. The first term of the approximation is equivalent to replacing the collection of atoms with a single atom, the second term approximates the error that remains after the first-term approximation by adding a function defined by two atoms, the third term employs four atoms to make the approximation even more precise, the fourth term uses eight atoms, and so on. The terms are referred to as the Monopole Moment, Dipole Moment and Quadrupole Moment respectively, to intuitively capture the order of approximation that has been applied in the chosen algorithm. These approximations can be captured in a constant number of terms that will be calculated per grid cell during construction of the hash structure. The surface visualisation then only has to read the terms of this function per neighbouring grid cell instead of all neighbouring atoms to approximate the value of the implicit function at its position.



The Mono- Di- and Quadrupole Moments visualized: the black dots are the approximating atoms inside the grid cell, the dashed dots the approximated original atoms of the grid cell and the white dot the surface particle in a neighbouring grid cell. Every term approximates the error left by the previous term, after which they are added together.

Figure 41

A detailed explanation is provided in [Appendix A]. The conclusion of this appendix is expressed in terms of [Equation 74] and [Equation 75]. While the first equation defines the calculation of the value of the implicit function at an arbitrary position by means of approximation, the second function establishes the number of terms that have to be (pre-) calculated per grid cell, in order to be able to evaluate the first equation. The amount of terms is a constant depending on the predetermined order of accuracy of the calculation.

In conclusion, the main reason to choose for Multipole Approximation is to reconstruct an implicit function at a position x by approximating implicit functions formed by the neighbouring cells while only having to retrieve a constant number of terms. With an order 2 approximation, one will need around 1 to 10 terms according to [Equation 75], while with our initial hash method B this amounts to m_a terms, where m_a is the average number of atoms per grid cell. The average number of atoms is 10 to 100 in most situations, but it is still possible that this number varies a lot for different grid cells.

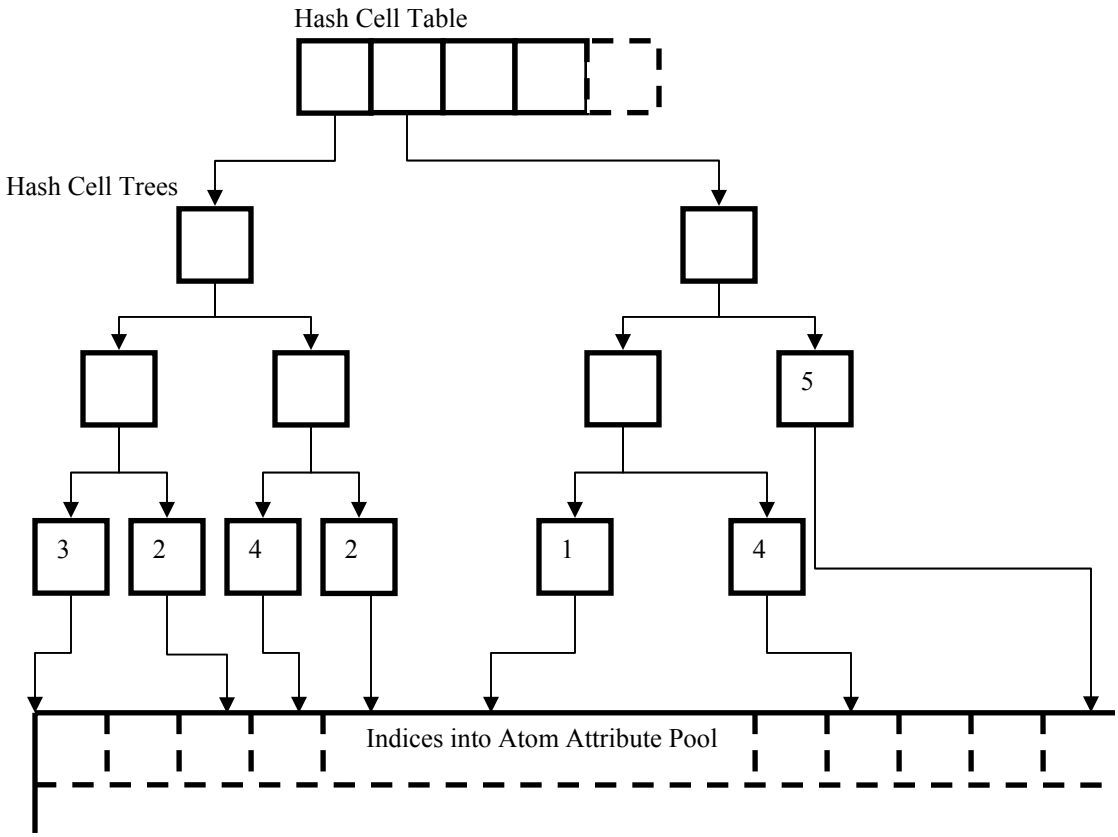
3.8.3 Applicability of FMM to the Particle Simulation

In section [3.8.2], a basic analysis of the data lookup requirements has already been made. The performance of the Fast Multipole Method for our hash structure described above while simulating the movement of the surface particles is for a large part dependent on the precision of the approximation. However, the applicability of FMM also depends on its usefulness for calculating our particle velocities in [Equation 36]. A closer look reveals that not only the value of F is required, but also its derivatives F^x for and F^q. Especially the latter derivative is expensive, for it requires derivatives of F over every atom $\boldsymbol{a}_j$. This means we get at least as much derivative terms as atoms, so it voids all performance advantages introduced by choosing FMM. To solve this, the term $F^q \bullet q$ should be approximated as well. Furthermore, calculating the derivative F^x and F^q requires derivation of a spherical coordinate over a Cartesian coordinate. When more derivatives are required to calculate aspects like curvature of the implicit function, even more terms should be precalculated, making the whole procedure very cumbersome and impractical. Also, it is debatable what the actual impact of the calculation of the approximation function of [Equation 74] is on the performance at a whole. The approximation function is quite large, so the extra instructions involved may offset the performance gained by the reduced latency from retrieving less data.

Another problem revolves around the choice of the Smoothing Kernel. The function of [Equation 59] approximated by FMM does not exhibit the properties required by a Smoothing Kernel. It has influence outside the radius, and the integral does not add up to 1. However, one might argue that the approximation only holds for particles inside or very close to an atoms influence radius, for only neighbouring grid cells use the approximation. Furthermore, the function might be a good approximation of a certain Smoothing Kernel, especially at medium to far distances.

The last problem caused by the choice for FMM is centered on the hash datastructure. So far, the discussion handily avoided this topic and talked about approximating the atoms inside a single grid cell. However, the hash data structure can map multiple grid cells to a single hash cell. Atoms from different grid cells cannot be approximated by a single function, because they may be located on different sides of an evaluation position. This can make it impossible to choose an origin for the approximation to which all atoms are closer than the evaluation position.

A solution to the last problem is to create a binary tree structure for each hash cell by which the grid cells are separated again. At every level of the tree, the current set of hash cells is intersected by a plane on the axis on which the diversity of grid cell positions is the largest. So when we have a collection of five grid cells with the coordinate set *{(1,3,9),(1,5,-9),(2,7,9),(8,2,-9),(0,4,1)}*, evaluation of the x-axis yields four different values, the y-axis yields five and the z-axis three, so the median of the y-axis is chosen for the next division. When one grid cell remains in a leaf, an index into the data table is created and the atoms are added to it. This is vaguely similar to building an OBB-tree or a KD-tree, see [VI] and [VII].



The Hash Cell Table does not directly address the Atom Attribute Pool; instead every cell has a tree splitting 3D space along a certain axis at every level and dividing the atoms of the cell according to the split. As soon as all remaining atoms belong to the same grid cell, an index into the Atom Attribute Pool is created.

Figure 42

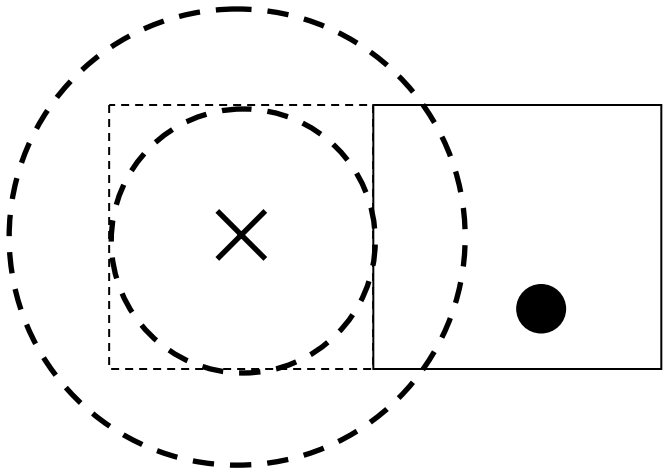
3.8.4 Adaptive Monopole Approximation

To overcome most of the problems faced by the FMM method, an approximation method is chosen based on the Monopole Moment of the Multipole Expansion theorem. If somehow a surface particle's neighbouring grid cell could be approximated by only a single atom with a position and a velocity, the implicit function and its derivatives defined by this single atom are easily calculated.

A trivial solution would be to take the centre of the neighbouring grid cell as the approximating atom's position, because it is a good approximation of the average of all atom positions in the grid as long as the grid is reasonably full. The mass of the atom should be equal to the sum of all atom masses in the grid cell and distributed around the atom by a Smoothing Kernel of choice with the standard atom radius.

This choice of approximation is easily discarded, because the standard atom radius would never cross the boundary of the grid cell, which has a size of two times the atom radius.

Therefore, a surface particle would never be influenced by any neighbouring particles. One could try to solve this problem by choosing a larger atom radius, but such a choice would lack a reasonable explanation, and become very subjective or even implementation dependent as a result.



The atom radius for an approximating atom is required to be larger than the grid cell, but establishing the right size is not trivial

Figure 43

A good observation in [Figure 43] would be to state that the position of the atom approximating the positions of atoms having influence on a surface particle should vary with the surface particle's position; the approximating atom position is adaptive. To see this, look at the atoms of the neighbour grid cell having an influence on the particle position; they are all located within the sphere of atom radius around the surface particle. This means that only the particles within the volume of the grid cell intersected by the sphere around the surface particle can have an influence on the surface particle, and therefore the average position of the atoms is different for each evaluation position.

Calculating a box-sphere intersection volume is expensive, so again an approximation routine has to be performed. Instead of a box-sphere intersection, a sphere-sphere intersection is calculated between the sphere around the centre of the neighbouring grid cell – the "grid cell sphere" – and the sphere around the surface particle. The intersection volume is always a lens with the following properties: the centre of the lens is located exactly in the middle of the line between the surface particle and the grid cell centre, and the volume can be calculated as follows:

$$\frac{1}{12\pi}\cdot(4r+d)\cdot(2r-d)^2$$

Equation 54

where r is the atom radius and d the distance between the neighbouring grid cell's centre and the surface particle.

The centre of the lens is a good approximation of the position of all atoms within the approximated intersection volume. The approximated mass is equal to the number of atoms within this volume. The calculation of that value is done according to the following equation:

$$N_{ia} = N_{ga} \cdot \frac{V_{gs}}{V_{gc}} \cdot \frac{V_{si}}{V_{gs}} = N_{ga} \cdot \frac{V_{si}}{V_{gc}}$$

Equation 55

where N_{ia} is the amount of atoms in the intersection volume, N_{ga} is the amount of atoms in the grid cell, V_{gc} is the volume of the grid cell, V_{gs} is the volume of the grid cell sphere and V_{si} is the volume of the sphere-sphere intersection.

The only problem that remains is the approximation of the atom velocities. Only the (approximated) amount of atoms within the intersection volume is known, but it is unknown which atoms of the neighbouring grid cell are within this volume. Therefore, the velocity of the approximating atom can only be approximated by the average of all atom velocities in the grid cell, calculated at construction of the hash structure and passed on to the surface particle visualization.

The result of this rather crude approximation is demonstrated in [Figure 45] and [Figure 45]; the method seems to work in dense areas and fall apart in areas of lower density. However, when looking at the effect in motion it becomes clear that the approximation is not good enough; the fluid surface does not stay intact for long at most positions as the fluid atoms move, causing the particles to fly around in space quite vehemently for most of the time.

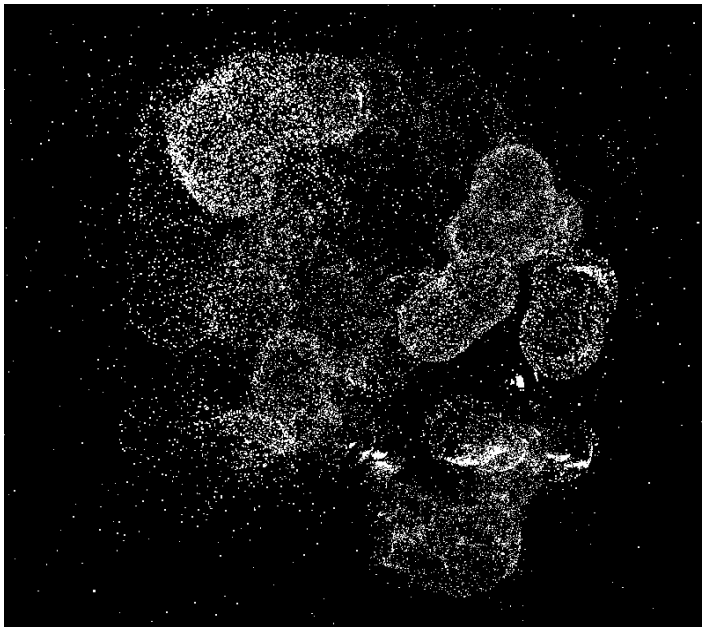


Image produced by the fluid surface visualisation with the Adaptive Monopole Approximation. Repulsion forces are disabled, and the surface particles are rendered as point sprites.

Figure 44

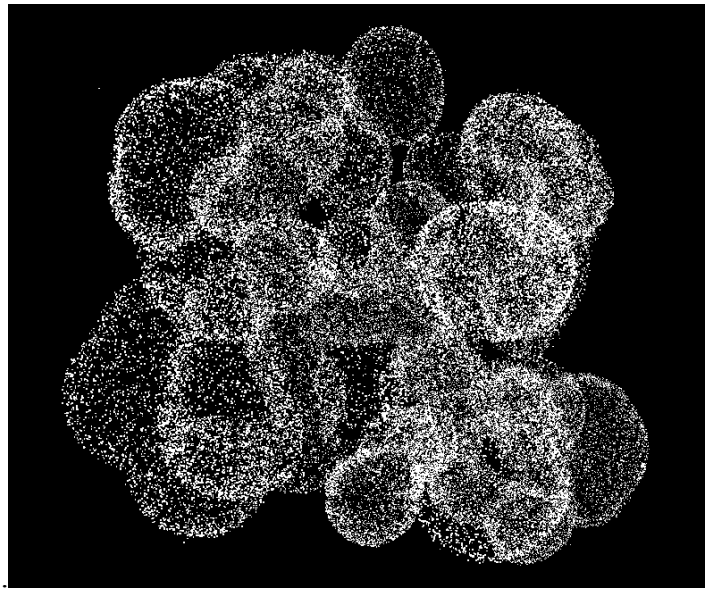


Image produced by the fluid surface visualisation without Adaptive Monopole Approximation. Repulsion forces are disabled, and the surface particles are rendered as point sprites.

Figure 45

4 Conclusions

4.1 Summary of results

In this work, an algorithm has been described for visualising a particle-based fluid with surface particles, under the constraint that it can be employed on commodity graphics hardware.

The first challenge encountered during the design of the algorithm, was to transport data from the fluid simulation to the visualisation in such a way, that the visualisation can reconstruct the fluid surface as efficiently as possible; i.e. with a minimum amount of data requested from the simulation and a minimum amount of computation performed on this data. This problem was solved in section [3.6.2] with the choice for a spatial hashing method. However, a spatial hash can be employed in more than one way, and every option has its own set of parameter choices. The optimal choice was investigated by section [3.7.2], and resulted in a surface reconstruction performance suitable for interactive applications.

After reconstruction of the fluid surface had been made possible, the simulated particles had to be constrained to the fluid surface. To this end, velocity constraints introduced by [II] were applied and augmented in section [3.3.1] with the choice for a specific "desired particle velocity", enabling particles to follow the fluid surface as it moves and changes.

Having accomplished the aforementioned, a method had to be found for distributing particles across the fluid surface. The first part of this method employs repulsion forces similar to those in [II], for local repulsion of surface particles. However, this method had to be adapted in order to be suitable for graphics hardware. A novel way to do this has been presented in section [3.6.4]. The second part of the distribution algorithm was constructed in order to improve the global distribution of particles, possibly between disconnected parts of the fluid surface. This method is based on the evaluation of density differences, making use of the same principle as the local distribution algorithm, in section [3.6.4].

The description of the visualisation ended with an investigation into methods for approximating the fluid surface by section [3.8], in order to improve the performance of reconstructing the fluid surface on the graphics hardware.

The result obtained by solving the problems mentioned in this section, is that a set of particles is optimally positioned each frame in order to represent the fluid surface as a whole. Furthermore, the simulation of the movement of these surface particles takes place entirely on the graphics hardware. This result can be used for rendering the fluid surface at every step of the simulation.

4.2 Future Work

The visualisation proposed in this work still leaves certain problems unsolved. Some are closely related to the discussed material, while other topics have not been touched on yet. The section will give an overview of the remaining problems, and tries to point into the direction of the solution where possible.

A problem closely related to those treated in this work, expands upon the distribution algorithm. As remarked before, the fluid surface that has to be visualised can consist of many disconnected components. These components can be quite small. An example could be the tiny holes forming in a big fluid "blob" as its fluid atoms move around. These holes can appear at any place inside the "blob", and they can be formed for very short amounts of time. However, it might be the case that no surface particles will ever be positioned on this new surface. In these cases no surface particle might be in the neighbourhood to identify the disconnected region of surface, for every surface particle might be on a local minimum of the implicit fluid surface function already – meaning that they already have a place on another part of fluid surface. So the mechanism constraining surface particles will not move any towards the newly created surface component. Because this surface component is disconnected from the rest and has no surface particles, the local and global repulsion algorithms will have no effect either; the local repulsion algorithm does not push surface particles onto disconnected surfaces, and the global repulsion algorithm has no comparison particle on the newly created surface to move another particle to. The result is that newly created disconnected surface components might never be visualised by surface particles. A solution to this problem might be to warp surface particles to random positions inside the influence areas of fluid atoms at specific time intervals. This way, newly created surfaces might be found after a small amount of time by randomly positioned fluid surface particles.

Another distribution-related problem is that of adaptive particle repulsion, as presented in [II]. This problem focuses on changing the number of particles on the fluid surface with their sizes under different conditions. One might consider a situation in which the fluid surface grows over time. To visualise this with the same level of detail at any point in time, surface particles have to be added or removed from the simulation. Otherwise, the size of the existing surface particles should increase or decrease. These situations can all be detected based on surface particle densities. One could also think of a level-of-detail-based algorithm, in which fluids further from the eye receive less surface particles that are enlarged, with closer surfaces receiving more and smaller particles. These changes in size and amount complicate the particle system and the distribution algorithm.

As made clear in section [3.6.4], the core algorithm for determining densities for surface particles is dependent on rendering the particles to a 2D viewport. This is the cause of a third problem. The viewport will usually be of limited size, which might create difficulties in local distribution when disconnected fluid components start to drift away from each other; many particles may map to a single screen pixel. In order to accurately distribute the particles over the surfaces of these fluids, it might be necessary to introduce multiple viewports for disconnected parts of the surface. Then, the different viewports

can be moved closer towards their respective parts of the fluid surface, resulting in better local distributions.

The last problem which remains for future research in particle-based fluid visualisation is the biggest one: the actual rendering of the fluid surface particles itself. This includes blending the surface particles correctly, texturing the fluid surface with regular textures, cube-maps or procedural textures, implementing reflection and refraction and determining the depth of the fluid at every pixel for volumetric effects. While many of these problems might be easily solved – the normal of the fluid surface can already be used for texturing, blending can be accomplished by using the depth buffer in one pass, and rendering front-most particles in the second – this work has not made any attempt to investigate what is possible and what is not. On top of that, focusing on rendering the surface particles will yield more spectacular results faster than slight optimisations of particle distribution algorithms.

A final remark has to be made about the choice for the fluid simulation model. The visualisation method described in this work is constructed on top of a particle-based fluid simulation using SPH, which has its own deficiencies when it comes to realistic simulation of fluids. A lot of fluid properties like vortices, wakes and circulation regions cannot directly be modelled by the current simulation model. However, even if the particle-based simulation method could somehow be extended to support these characteristics, the visualisation method would not have to change to be able to visualise it. As long as the position of the fluid atoms is the only quantity required to define the fluid surface and the simulation can send these with their corresponding velocities to the visualisation, any fluid simulation model suffices. This also holds when looking in the opposite direction; the described fluid visualisation can be used in less complex scenarios as well. All kinds of objects consisting of metaballs – even those with very simple behaviour – can be visualised with the described algorithm. Think of animated characters in games consisting of fluid atoms, with custom animations representing the movement of a human or animal. The choice for visualising a fluid simulation is merely interesting because of its upcoming application in computer games. There, rigid body simulation is performed frequently nowadays, but dynamically simulated blobby objects are still uncommon.

5 Appendix A

5.1 Multipole Expansion

Naively, evaluating the implicit function defined by a set of atoms at position $\boldsymbol{x}$ can be generalised to the following function, according to [Equation 43]:

$$f(\boldsymbol{x},\boldsymbol{q}) = \sum_{j=1}^{m} c \cdot K\left(\boldsymbol{x} - \boldsymbol{a}_j\right)$$

Equation 56

for K a particular kernel function, c the constant atom mass, $\boldsymbol{q} = (\boldsymbol{a}_1,..,\boldsymbol{a}_m)$ and atom positions $\boldsymbol{a}_1..\boldsymbol{a}_m$. One should realise that in this case, q represents the atoms in a single neighbour grid cell, and $\boldsymbol{x}$ is the position of a surface particle under influence of these atoms. For every such surface particle, a part of its total implicit function at position $\boldsymbol{x}$ is defined by [Equation 56]; requiring every particle to provide exactly the same parameter $\boldsymbol{q}$ but varying $\boldsymbol{x}$.

The Fast Multipole Method aims to eliminate the dependency between every $\boldsymbol{x}$ and $\boldsymbol{a}_j$ from [Equation 56], such that a part of the equation can be calculated ahead of time. In fact, if we could write [Equation 56] more or less as follows:

$$f(\boldsymbol{x},\boldsymbol{q}) \approx \sum_{j=1}^{m}\left(c \cdot K_1\left(\boldsymbol{a}_j\right) \cdot K_2\left(\boldsymbol{x}\right)\right) = K_2\left(\boldsymbol{x}\right) \cdot \sum_{j=1}^{m}\left(c \cdot K_1\left(\boldsymbol{a}_j\right)\right)$$

Equation 57

then, because all $K_1(\boldsymbol{a}_j)$ would yield the same value for every surface particle under influence of $\boldsymbol{q}$, this part can be can be calculated before evaluating the implicit function at every surface particle position. So, for every grid cell G with the atom positions $\boldsymbol{a}_1..\boldsymbol{a}_m$ encapsulated by it, the value of the expression

$$f_G = \sum_{j=1}^{m}\left(c \cdot K_1\left(\boldsymbol{a}_j\right)\right)$$

Equation 58

should be calculated and stored per grid cell, after which the surface visualisation can query this value for every neighbour cell of a particle. This would reduce the amount of data lookups and computations to evaluate the implicit function at neighbour cells to $O(1)$ complexity.

Of course, the question that arises is to find a good choice for K_1 and K_2.

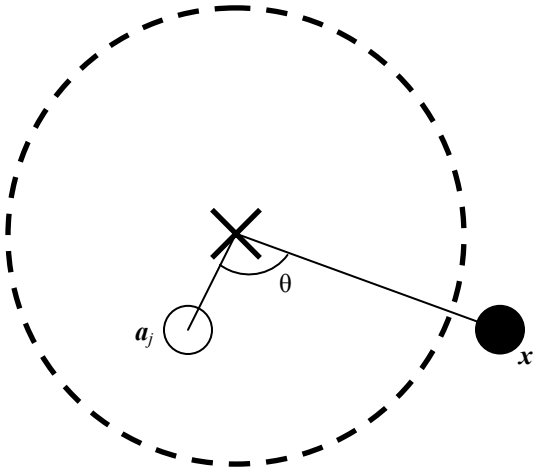
When looking at [Equation 56] again and combining it with the implicit function from [Equation 43], we could state that $K(\boldsymbol{x}-\boldsymbol{a}_j) = W(\boldsymbol{x}-\boldsymbol{a}_j)-B/m$. However, for simplicity and the applicability of the Fast Multipole Method, the following is assumed:

$$K\left(\boldsymbol{x}-\boldsymbol{a}_j\right)=\frac{1}{\left|\boldsymbol{x}-\boldsymbol{a}_j\right|}$$

Equation 59

Of course this discrepancy has to be resolved in a later stage, especially because this function exhibits few of the properties of a Smoothing Kernel.

The advantage of defining $K(\boldsymbol{x}-\boldsymbol{a}_j)$ according to [Equation 59], is that the function can now be approximated by a method called Multipole Expansion. Say, the fluid atom $\boldsymbol{a}_j$ and evaluation position $\boldsymbol{x}$ are located around an origin in spherical coordinates, with $\boldsymbol{a}_j$ closer to the origin than $\boldsymbol{x}$:



A single fluid atom (white) with surface particle (black) around the origin denoted by a cross, with angle θ

Figure 46

As outlined in [V], it is possible to rewrite [Equation 59] with a rule from trigonometry:

$$\left|\boldsymbol{x}-\boldsymbol{a}_j\right|^2=\left|\boldsymbol{a}_j\right|^2+\left|\boldsymbol{x}\right|^2-2\left|\boldsymbol{a}_j\right|\left|\boldsymbol{x}\right|\cos\theta$$

Equation 60

such that

$$\frac{1}{\left|\boldsymbol{x}-\boldsymbol{a}_j\right|}=\frac{1}{\sqrt{\boldsymbol{x}^2-2\left|\boldsymbol{a}_j\right|\left|\boldsymbol{x}\right|\cos\theta+\boldsymbol{a}_j{}^2}}=\frac{1}{\left|\boldsymbol{x}\right|}\cdot\frac{1}{\sqrt{1-2\frac{|a_j|}{|x|}\cos\theta+\frac{|a_j|^2}{|x|}}}=\frac{1}{\left|\boldsymbol{x}\right|}\cdot\frac{1}{\sqrt{1-2vu+v^2}}$$

$$\text{where } \mathrm{v}=\frac{|a_j|}{|x|} \text{ and } \mathrm{u}=\cos\theta$$

Equation 61

Rewriting [Equation 59] by this equation does not appear useful, but the last term

$$g(u,v) = \frac{1}{\sqrt{1-2vu+v^2}}$$

Equation 62

is a generating function for Legendre polynomials P_s:

$$g(u,v) = \sum_{s=0}^{\infty} P_s(u) v^s$$

Equation 63

a series approximation convergent for $|v| < 1$, which is in line with our requirement that $|\boldsymbol{a}_j| < |\boldsymbol{x}|$. So finally, [Equation 59] can be rewritten:

$$K(\boldsymbol{x}-\boldsymbol{a}_j) = \frac{1}{|\boldsymbol{x}-\boldsymbol{a}_j|} = \frac{1}{|\boldsymbol{x}|} \cdot \sum_{s=0}^{\infty} P_s(\cos\theta) \left(\frac{|\boldsymbol{a}_j|}{|\boldsymbol{x}|} \right)^s = \sum_{s=0}^{\infty} \left(|\boldsymbol{a}_j|^s P_s(\cos\theta) \right) \left(\frac{1}{|\boldsymbol{x}|^{s+1}} \right)$$

Equation 64

This equation is already much more in line with the idea presented in the form of [Equation 57]; our independent functions K_1 and K_2 can be identified within the first pair of parenthesis and the second pair respectively. Still, while K_2 is dependent on $|\boldsymbol{x}|$ alone, K_1 is dependent on $|\boldsymbol{a}_j|$ as well as θ. The angle θ depends on both $|\boldsymbol{a}_j|$ and $|\boldsymbol{x}|$, which means the goal of an independent K_1 and K_2 has not been reached yet.

On a sidenote, the Legendre polynomials introduced in [Equation 63] can be defined in multiple ways. Two of these are the recurrence relation:

$$(2n+1)uP_s(u) = (n+1)P_{s+1}(u) + nP_{s-1}(u)$$

Equation 65

and Rodrigues' formula:

$$P_s(u) = \frac{1}{2^s n!} \frac{d^s}{\partial u^s} \left(u^2 - 1\right)^s$$

Equation 66

as listed in [V].

For the purpose of this discussion only $P_0(u) = 1$ and $P_1(u) = u$ are required.

Continuing the analysis of [Equation 64], it is obvious that it is impractical to calculate the infinite sum of terms containing the Legendre polynomials. Therefore, it is useful to gain some understanding into the meaning of each of the individual terms.

If $K(\boldsymbol{x}-\boldsymbol{a}_j)$ is approximated only by the term in which $s = 0$, an expression is obtained containing P_0, so the whole equation is reduced to just $|\boldsymbol{x}|^{-1}$. Looking at the implicit function from [Equation 56], the approximation yields the following expression:

$$f(\boldsymbol{x},\boldsymbol{q}) = \sum_{j=1}^{m} c \cdot K\left(\boldsymbol{x}-\boldsymbol{a}_j\right) = \sum_{j=1}^{m} \frac{c}{|\boldsymbol{x}|} = m\frac{c}{|\boldsymbol{x}|}$$

Equation 67

which is equivalent to positioning a single point mass at the origin of the spherical coordinate frame with a mass equal to the total mass of all atoms 1 to m. This approximation of the implicit function at a single location is called the "Monopole Moment". Notice that the Monopole Moment is only dependent on $|\boldsymbol{x}|$, unlike [Equation 64]. If so desired, one could use this equation to approximate the implicit function defined by these atoms at any position, as long as the position is further from the origin than any of the atoms.

The second term in [Equation 64] would be the one containing P_1, and when the equation is then substituted into [Equation 56], the implicit function becomes:

$$f(\boldsymbol{x},\boldsymbol{q}) = \sum_{j=1}^{m} c \cdot K\left(\boldsymbol{x}-\boldsymbol{a}_j\right) = \frac{1}{|\boldsymbol{x}|^2} \sum_{j=1}^{m} \left(c \cdot |\boldsymbol{a}_j| \cdot \cos\theta\right)$$

Equation 68

which is called the "Dipole Moment". In electrodynamics, the constant mass c is replaced by an atom charge q_j for every atom, so in the case of two atoms with $q_1 = -q_2$ the Monopole Moment becomes zero and the Dipole Moment defines the approximated electrostatic field of the two point charges.

At every step, the number of atoms approximated by the moment is multiplied by two, so the third moment is called "Quadrupole Moment", the fourth the "Octupole Moment", and so on.

The maximum moment determines the amount of precision in the final approximation. Since there are only a limited number of atoms per grid cell – 11.7 in the case of hash method B – it would not be of significant benefit to choose approximations based on more than 16 atoms.

The problem that still has to be solved is the dependency of the two components in the sum from [Equation 64]. It is especially desirable to replace the term $P_s(\cos\theta)$ with components that are not dependent on both $\boldsymbol{x}$ and $\boldsymbol{a}_j$. By using spherical harmonics, it is

possible to do exactly that. According to [V], the Legendre polynomial can be written in terms of spherical harmonics in the following way:

$$P_s(\cos\theta) = \sum_{t=-s}^{s} Y_s^{-t}(\boldsymbol{a}_i) \cdot Y_s^t(\boldsymbol{x})$$

Equation 69

where Y is the spherical harmonics term:

$$Y_s^t(\boldsymbol{y}) = (-1)^t \cdot \sqrt{\frac{2s+1}{4\pi}\frac{(s-|t|)!}{(s+|t|)!}} \cdot P_s^t \cdot (\cos\theta_y) \cdot e^{i\phi_y t}$$

Equation 70

with θ_y and φ_y the polar and azimuthal angles of $\boldsymbol{y}$ respectively, and i the imaginary number for which $i^2=-1$. An extension of our standard definition of the Legendre polynomial is required as well to be able to evaluate the previous equation:

$$P_s^t(u) = (1-u^2)^{\frac{t}{2}} \frac{d^t}{\partial u^t} P_s(u)$$

Equation 71

For $t = 0$, [Equation 71] yields $P_s(u)$. According to [V], a recursive definition can also be obtained; because of the lack of importance in this discussion I have chosen to omit it.

According to [Equation 69], it is finally possible to break down the equation into independent terms. However, this comes at a cost: [Equation 70] introduces an imaginary component, denoting that $P_s(\cos\theta)$ is not independent in real space, only in complex space.

Continuing our derivation of $K(\boldsymbol{x}-\boldsymbol{a}_j)$ from [Equation 64], the independent terms are clearly distinguishable:

$$K(\boldsymbol{x}-\boldsymbol{a}_j) = \sum_{s=0}^{\infty}\left(|\boldsymbol{a}_j|^s \cdot P_s(\cos\theta) \cdot \frac{1}{|\boldsymbol{x}|^{s+1}}\right) =$$

$$\sum_{s=0}^{\infty}\left(|\boldsymbol{a}_j|^s \cdot \left(\sum_{t=-s}^{s}\left(Y_s^{-t}(\boldsymbol{a}_j) \cdot Y_s^t(\boldsymbol{x})\right)\right) \cdot \frac{1}{|\boldsymbol{x}|^{s+1}}\right) =$$

$$\sum_{s=0}^{\infty}\sum_{t=-s}^{s}\left(\left(|\boldsymbol{a}_j|^s \cdot Y_s^{-t}(\boldsymbol{a}_j)\right) \cdot \left(\frac{Y_s^t(\boldsymbol{x})}{|\boldsymbol{x}|^{s+1}}\right)\right)$$

Equation 72

Plugging this into the implicit function, the independent parts can be separated:

$$f(\boldsymbol{x},\boldsymbol{q}) = \sum_{j=1}^{m} c \cdot K\left(\boldsymbol{x} - \boldsymbol{a}_j\right) =$$

$$\sum_{j=1}^{m} c \cdot \left(\sum_{s=0}^{\infty} \sum_{t=-s}^{s} \left(\left(\left|\boldsymbol{a}_j\right|^s \cdot Y_s^{-t}\left(\boldsymbol{a}_j\right) \right) \cdot \left(\frac{Y_s^t(\boldsymbol{x})}{\left|\boldsymbol{x}\right|^{s+1}} \right) \right) \right) =$$

$$\sum_{s=0}^{\infty} \sum_{t=-s}^{s} \left(\left(\sum_{j=1}^{m} c \cdot \left|\boldsymbol{a}_j\right|^s \cdot Y_s^{-t}\left(\boldsymbol{a}_j\right) \right) \cdot \left(\frac{Y_s^t(\boldsymbol{x})}{\left|\boldsymbol{x}\right|^{s+1}} \right) \right)$$

Equation 73

where *j* iterates over all atoms, *s* iterates over all degrees of approximation and *t* iterates over all spherical harmonics components.

This expression is usually simplified to the following notation:

$$f(\boldsymbol{x},\boldsymbol{q}) = \sum_{s=0}^{\infty} \sum_{t=-s}^{s} \left(M_s^t(\boldsymbol{q}) \cdot \left(\frac{Y_s^t(\boldsymbol{x})}{\left|\boldsymbol{x}\right|^{s+1}} \right) \right)$$

where

$$M_s^t(\boldsymbol{q}) = \sum_{j=1}^{m} c \cdot \left|\boldsymbol{a}_j\right|^s \cdot Y_s^{-t}\left(\boldsymbol{a}_j\right)$$

Equation 74

The terms of *M* are dependent on $\boldsymbol{q}$ only, and the remaining part is only dependent on $\boldsymbol{x}$. Therefore, the terms of *M* can be calculated per grid cell and read back at the evaluation of the implicit function at particle locations.

It is clear now that for the approximation level *s* of order *n = s+1*, *s*2+1* terms of *M* will have to be calculated. So for the Monopole Moment one term M_0^0 is required, the Dipole Moment requires three terms $\{M_1^{-1}, M_1^0, M_1^1\}$, the Quadrupole Moment five, and so on. The full approximation of the implicit function then requires the following amount of terms *M*:

$$\sum_{s=0}^{n-1} (2s+1) = O\left(n^2\right)$$

Equation 75

with *n* the number of approximation levels.

As noted before, these terms consist of a real and an imaginary component. So at first glance, to approximate the function up until the dipole moment, one would need 4*2 = 8 scalar values. However, [V] shows that M_s^{-t} can be derived from M_s^t, and [Equation 70] combined with [Equation 74] prove that M_s^0 only has a real component. Therefore, the number of required scalar values for approximation up to the Dipole Moment only requires 8-2-2 = 4 scalar values, which can be retrieved by a single texture lookup. Calculating the Quadrupole Moment requires 5 scalar values, bringing the total up to 9 scalar values. This results in three texture lookups already. So the level of approximation will have to be carefully considered, even if the only action performed is the evaluation of the basic approximation function.

6 References

[I] Matthias Müller, David Charypar, Markus Gross
Particle-Based fluid simulation for interactive applications
Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 154 – 159, 2003

[II] Andrew P. Witkin, Paul S. Heckbert
Using particles to sample and control implicit surfaces
Proceedings of the 21st annual conference on Computer graphics and interactive techniques, 1994

[III] M. Teschner, B. Heidelberger, M. Müller, D. Pomeranets, M. Gross
Optimized Spatial Hashing for Collision Detection of Deformable Objects
Proceedings of Vision, Modeling, Visualization VMV'03, 2003

[IV] T. Amada, M. Imura, Y. Yasamuro, Y. Manabe, K. Chihara
Particle-Based Fluid Simulation on GPU
ACM Workshop on General-Purpose Computing on Graphics Processors and SIGGRAPH 2004 Poster Session, 2003

[V] Alexander T. Ihler
An Overview of Fast Multipole Methods
http://www.ics.uci.edu/~ihler/papers/ihler_area.pdf, 2004

[VI] S. Gottschalk, M.C. Lin, D. Manocha
OBBTree: A Hierarchical Structure for Rapid Interference Detection
Computer Graphics (SIGGRAPH '96 Proceedings), pages 171—180, 1996

[VII] Tim Foley, Jeremy Sugerman
KD-Tree Acceleration Structures for a GPU Raytracer
Proceedings of the ACM SIGGRAPH/Eurographics conference on Graphics hardware, pages 15 – 22,
2005

[VIII] Lutz Latta
Building a Million Particle System
http://www.2ld.de/gdc2004/, 2004

[IX] Mathieu Desbrun, Marie Paule Cani
Smoothed Particles: A new paradigm for animating highly deformable bodies
Eurographics Workshop on Computer Animation and Simulation (EGCAS), pages 61 – 76, 1996

[X] Jos Stam, Eugene Fiume
Depicting fire and other gaseous phenomena using diffusion processes
Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, pages 129 – 136, 1995

[XI] J. P. Morris
Simulating surface tension with Smoothed Particle Hydrodynamics
International Journal for Numerical Methods in Fluids, pages 333 – 353, 2000

[XII] M. J. Harris
Fast Fluid Dynamics Simulation on the GPU
GPU Gems, Chapter 38, 2004

[XIII] Helwig Hauser, Robert S. Laramee, Helmut Doleisch
State-of-the-Art Report 2002 in Flow Visualisation
TR-VRVis-2002-003, 2002

[XIV] William E. Lorensen, Harvey E. Cline
Marching cubes: A high resolution 3D surface construction algorithm
Proceedings of the 14th annual conference on Computer graphics and interactive techniques, pages 163 – 169, 1987

[XV] V. Pascucci
Isosurface Computation Made Simple: Hardware Acceleration, Adaptive Refinement, Tetrahedral Stripping
Joint Eurographics - IEEE TVCG Symposium on Visualization (VisSym), pages 293 – 300, 2004

[XVI] GPGPU: General Purpose Computation on Graphics Hardware
http://www.gpgpu.org/s2005/FullCourseNotes.pdf